BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

ANOMALY DETECTION IN IEC 61850 COMMUNICATION

DETEKCE ANOMÁLIÍ V PROVOZU IEC 61850

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **DANIELA PEŠKOVÁ**
AUTOR PRÁCE

SUPERVISOR **Ing. PETR MATOUŠEK, Ph.D., M.A.**
VEDOUCÍ PRÁCE

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav informačních systémů (UIFS) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Pešková Daniela**
Program: Informační technologie
Název: **Detekce anomálií v provozu IEC 61850**
Anomaly Detection in IEC 61850 Communication
Kategorie: Počítačové sítě

Zadání:

1. Seznamte se s průmyslovou komunikací IEC 61850 (protokoly MMS a GOOSE).
2. Prostudujte metody pro detekci anomálií v průmyslové komunikaci. Zaměřte se popis komunikace pomocí jazyků a statistickou analýzu.
3. Navrhněte způsob modelování komunikace pomocí automatů s využitím pravděpodobnosti, případně statistických údajů. Algoritmus implementujte jako nástroj, který čte příznaky (features) z odchycené komunikace a automaticky vytváří model.
4. Popište způsob učení se (vytváření automatů) na základě trénovací množiny a klasifikace provozu (detekce anomálií). Navržený postup otestujte na různých datových sadách.
5. Shrňte získané výsledky a diskutujte využití a možné rozšíření.

Literatura:

- MATOUŠEK Petr. Description of IEC 61850 Communication. FIT-TR-2018-01, Brno: Fakulta informačních technologií VUT v Brně, 2018.
- Kwon, Y.; Lee, S.; King, R.; Lim, J.I.; Kim, H.K. Behavior Analysis and Anomaly Detection for a Digital Substation on Cyber-Physical System. *Electronics*2019, *8*, 326.
- Matoušek, Petr, Ryšavý, Ondřej, Grégr, Matěj, Havlena, Vojtěch: Flow Based Monitoring of ICS Communication in the Smart Grid, 2020, Journal of Information Security and Applicatons. Submitted.
- Yoo, H., Shon, T. Novel Approach for Detecting Network Anomalies for Substation Automation based on IEC 61850. *Multimed Tools Appl* 74,303-318 (2015). https://doi.org/10.1007/s11042-014-1870-0

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Petr, Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 2. listopadu 2020

Abstract

This thesis deals with anomaly detection in industrial communication IEC 61850. It studies using various statistical analysis methods and probabilistic automata for creating communication profiles and its accuracy while detecting anomalies.

Abstrakt

Táto práca sa zaoberá detekciou anomálií v priemyslovej komunikácii IEC 61850. Práca skúma použitie rôznych štatistických metód a pravdepodobnostných automatov na vytvorenie profilu komunikácie v sieti a jej presnosť pri rozpoznávaní anomálií.

Keywords

anomaly detection, IEC 61850, MMS, GOOSE, statistical analysis, probabilistic automata

Klíčová slova

detekcia anomálií, IEC 61850, MMS, GOOSE, štatistická analýza, pravdepodobnostné automaty

Reference

PEŠKOVÁ, Daniela. *Anomaly Detection in IEC 61850 Communication.* Brno, 2021. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Petr Matoušek, Ph.D., M.A.

Anomaly Detection in IEC 61850 Communication

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Petr Matoušek, Ph.D., M.A. The supplementary information was provided by Ing. Vojtěch Havlena. I have listed all the literary sources, publications, and other sources, which were used during the preparation of this thesis.

.....
Daniela Pešková
May 18, 2021

Acknowledgements

I would like to thank Ing. Petr Matoušek, Ph.D., M.A. for his guidance, knowledge, and help he provided during my work. I would also like to thank Ing. Vojtěch Havlena for providing the tool partially used in the thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The considerable increase of threads to network users, including critical infrastructures using the power grid has shown the need to increase its security. Currently, smart grid protection includes IDS (Intrusion Detection Systems), IPS (Intrusion Prevention Systems), and firewalls. These are usually deployed on the edge of the network, protecting it against external threats. For internal threats, an anomaly detection system that identifies possible attacks by monitoring data is necessary. In this thesis, a combination of statistical analysis and probability automata is used to demonstrate anomaly detection on IEC 61850 communication.

IEC 61850 is a global standard for substation automation. It defines communication between devices using abstract objects. These abstract models can be then mapped to various protocols. Some current mappings related to this thesis are to Manufacturing Message Specification (MMS) and Generic Object Oriented Substation Event (GOOSE) protocols. GOOSE communication tends to be stable, and we can observe not changing easily describable patterns. While MMS packet transmission is more complex and therefore more problematic to describe, repeated patterns can still be recognized. Because each protocol is built on a contrasting base, different anomaly detection methods will be used.

In this thesis, we are going to try to understand industrial communication by analyzing its patterns. GOOSE protocol will be studied using statistical analysis, in particular data will be represented by arithmetic mean and variance, box plot, and confidence interval. MMS communication will be learned and then detected using the power of probabilistic automata.

The tool we created for this purpose is divided into two phases. In the learning phase, the tool tries to recognize patterns of communication and to create descriptions or metrics of them. In the detecting phase, smaller portions of data are compared to these learned models. The result of the thesis should be to determine whether or not our approach is usable in industrial communication and to understand how various methods perform on data.

This thesis is structured as follows. Chapter 2 reviews methods used for anomaly detection. Chapter 3 presents industrial communication in IEC 61850, gives details about GOOSE and MMS protocols, as well as about general definition of abstract mapping objects in the standard. Chapter 4 gives an overview of the profiling tool created for purpose of this thesis. It shows how the tool is structured and implemented, and also it gives details about how different phases of detection work together and how the data sets are used for experiments. Chapter 5 describes experiments on data and the results of approaches we used. The last chapter concludes this paper.

Chapter 2

Detection Methods

Finding hidden information in given data is essential for detecting anomalies in the time series. This process is often called *data mining*. There are many different methods used to perform data mining tasks. Chapter 2 will offer a brief background on detection methods used by profiling tool.

Chapter is divided into two sections. Statistics and probability described in the first section can be used for simple protocols with a tendency to stable and unchanging behavior. For our thesis, we consider Generic Object Oriented Substation Event protocol or GOOSE. Probabilistic automata will be better used with protocols where communication between nodes in the network is also stable and communication patterns are periodical, but they can change from time to time, such as Manufacturing Message Specification, or MMS.

2.1 Statistics and Probability

Statistics will be used to help us find similarities between collections of statistical data. It is possible to find representatives of each collection and then compare them using various methods.

The basic concept behind this approach is simple. Each method uses different representatives. These are computed separately for each module. E.g. if we have one training module and one detecting module, a *statistical method* will be used for each module independently. After finding our representatives, we can then define the *probability* with which the ones from the detecting module are similar to the ones from the learning module.

2.1.1 Arithmetic Mean and Variance

Arithmetic mean is a basic concept in statistics, that can provide an abstraction of given data using central tendency measures. In fact, according to [1] arithmetic mean is the most effective numerical measure of the center of a set of data. We define it as the sum of all measured elements divided by the size of a set of all measured elements, such as:

$$\overline{x} = \frac{1}{n}(\sum_{i=1}^{n} x_i) \tag{2.1}$$

where n is the count of all numbers in collection and x_i is one numerical value from the collection.

This method simply gives us an average value, which can be then used as a representative of the given collection.

Variance can be easily combined with arithmetic mean. It measures an interval in which numbers can spread out from their average value. We define it as

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \overline{x})^2 \tag{2.2}$$

where $\overline{x}$ represents the arithmetic mean of a given collection.

Although mean and its variance are useful tools for describing a data set, they have one major problem. Because of their sensitivity to outliers, even a small amount of extreme values can corrupt the representatives. For example, the average BMI (body mass index) of a group of people can be significantly pushed up by a few heavier members.

Despite the fact that this is exactly what we need when detecting threats, its sensitivity can be problematic when calculating data from the learning module. This is why it is useful to sometimes chop off values at the high and low extremes, or use more advanced methods.

2.1.2 Box Plot

There are some popular techniques for displaying data graphically. One of them is the box plot [1]. It illustrates several features of data using quartiles.

Quartiles can be described as values dividing data into quarters. This means each of the four parts contains 25% of the data.

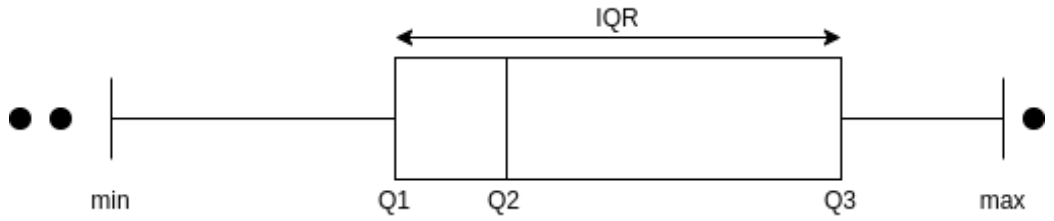

Figure 2.1: Box plot example

As we can see in Figure 2.1, distribution consists of (written in the order) *minimum*, *lower quartile* (Q1), *median* (Q2), *higher quartile* (Q3), and *maximum*.

The length of the box in the center of the figure represents the interquartile range or IQR. The line within the box is median and the lines extending outside the box, also called whiskers, are **1.5xIQR** of the quartiles. IQR is defined as

$$IQR = Q3 - Q1 \tag{2.3}$$

Beyond the whiskers, potential outliers are shown. These values are extremely low or high observations that do not fit into whiskers.

To roughly determine the first and the third quartiles, it is necessary to sort given data. Then:

- Q2 is the median of the data.
- Q1 is the median of the data on the left of the median.
- Q3 is the median of the data on the right of the median.

Box plots can be used in a comparison of sets of data to show if data is symmetrical and to point out potential outliers. Despite this method is mainly used graphically, it is possible to use it to reach our goal as shown in Chapter 4.

2.1.3 Confidence Interval

Another approach to statistical analysis is to use confidence interval [1]. This means to determine a range of values within which the true parameter value should fall in with some degree of uncertainty.

Whereas the confidence interval provides an estimation, the sample of data does not necessarily include its true value. Because of this, the confidence level is given by the user. This means that if we say the *confidence level* is 95%, when the estimation process is repeated over and over, ensuring 95 % of calculated intervals will contain the true value.

We have decided to use the three-sigma rule, ensuring a 99 % confidence level.

Standard Deviation

Standard deviation, σ, is an algebraic measure computed as the square root of the variance, σ^2. Standard deviation measures the spread around the mean. Only when all observations have the same value, σ equals zero.

Three-sigma Rule

Assuming that the distribution of given data is normal, the three-sigma rule can set the upper and lower control limits. This rule is a statistical calculation where data within three *standard deviations*, or σ, from a *mean*, are considered confidence interval.

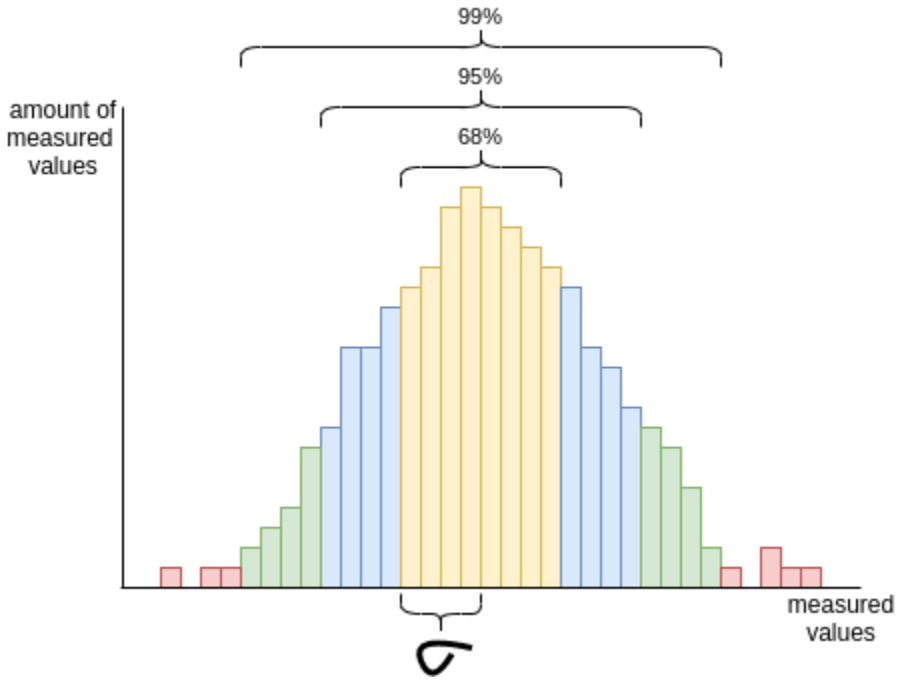

Figure 2.2: Three-sigma rule

As shown in Figure 2.2, 68%, 95%, and 99% is the confidence level of the interval within one, two, and three standard deviations from a mean.

However, a high confidence rate comes with consequences. Figure 2.2 shows, that the higher the confidence, the bigger the confidence interval. This means some threads could stay hidden in the interval during detection.

We are able to detect anomalies by measuring standard deviations of chunks of data and comparing them.

2.2 Probabilistic Automata

If a language is a set of strings, the string either belongs to a language or does not. Because of this, a distribution over the set of all strings can be defined. This distribution can be then used to predict the next symbol for a given prefix or can find the most probable string corresponding to a pattern.

The finite state machines used to recognize strings are called automata. The next state is chosen based only on the previous state. This procedure can be made deterministic by allowing only one action to be possible at each step. The automata usually has three types of states - *accepting*, *rejecting*, and *neutral*. The first two are also called *final* states.

Probabilistic automata, also called Markov Chain, is a non-deterministic automaton with probabilities assigned to its transition function. In [6] and [8] it is defined as follows:

Definition 2.2.1. *Probabilistic automaton* (PA) *is a tuple* $A = (\Sigma, Q, \delta, \mathbb{I}, \mathbb{F})$ *where*

- Σ *is an alphabet*,
- Q *is a finite set of states*,
- $\delta : Q\Sigma Q \to Q \cap [0, 1]$ *is a (total) transition function assigning probabilities to each transition. If* $\delta(q1, a, q2) = 0$ *the transition from* $q1$ *to* $q2$ *via symbol* a *is not actually there*,
- $\mathbb{I} : Q \to Q \cap [0, 1]$ *is a mapping assigning initial-state probabilities, and*
- $\mathbb{F} : Q \to Q \cap [0, 1]$ *is a mapping assigning final-state probabilities.*

The following conditions hold:

$$\sum_{q \in Q} \mathbb{I}(q) = 1,$$

and $\forall q \in Q$*:*

$$\mathbb{F}(q) + \sum_{a \in \Sigma, r \in Q} \delta(q, a, r) = 1.$$

Definition 2.2.2. *Probabilistic automaton* $A = (\Sigma, Q, \delta, \mathbb{I}, \mathbb{F})$ *is called deterministic, if*

- $\exists q \in Q : \mathbb{I}(q) = 1$,
- $\forall q \in Q, \forall a \in \Sigma : |\{r | \delta(q, a, r)\}| = 1$.

If a word w is a sequence of symbols in the alphabet, its probability can be found as follows. Let $w = a_1 a_2 ... a_n \in \Sigma^*$[6] be a word. A trace π of the word w is a sequence $\pi = (q_0, a_1, q_1)...(q_{n-1}, a_n, q_n)$ where $\delta(q_{i-1}, a_i, q_i) > 0$, $\mathbb{I}(q_0) > 0$, and $\mathbb{F}(q_n) > 0$ for $1 \leq i \leq n$. Informally, a trace is a path through PA via the word w ending in a state with non-zero probability.

Let Π_w be a set of all traces of a word w. Probability of the path π is then given as $\mathbb{P}_A(\pi) = \mathbb{I}(q_0) * (q_0, a_1, q_0) * ... * (q_{n-1}, a_n, q_n) * \mathbb{F}(q_n)$.

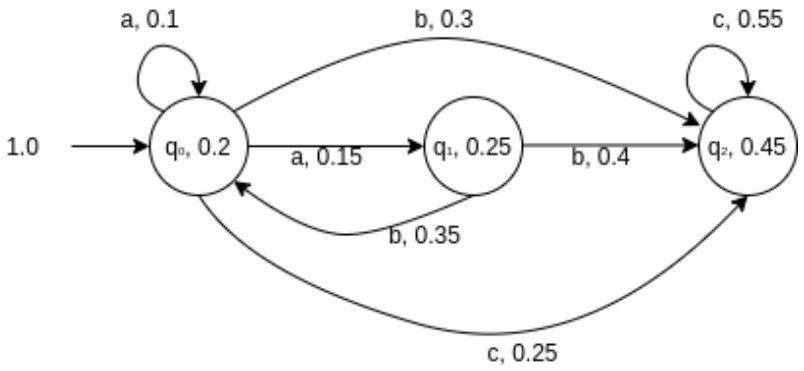

Figure 2.3: Example of simple probabilistic automaton

The probability of the word w is then given as a sum of all probabilities of traces for a given word w, such that $\mathbb{P}_A(w) = \sum_{\pi \in \Pi_w} \mathbb{P}_A(\pi)$. A word w is accepted by a PA if $\mathbb{P}_A(w) > 0$.

Consider PA from figure 2.3. States are labeled with a state name and the accepting probability. Transitions are labeled with a symbol and the probability taking this transition. Following the definitions, probability of word abc is following: $\mathbb{P}_A(abc) = 1.0 * 0.1 * 0.3 * 0.55 * 0.45 + 1.0 * 0.15 * 0.35 * 0.25 * 0.45 + 1.0 * 0.15 * 0.4 * 0.55 * 0.45$.

Learning Probabilistic Automata

Steps for learning PA were copied from [6]. They are describing the algorithm ***Alergia***. The algorithm follows three ideas: a predefined ordering of the states, a compatibility test, and a merging operation [8]. It takes as an input a multiset of strings S and outputs a deterministic probabilistic automaton corresponding to S. It proceeds in the following steps:

1. Create a prefix tree with number-labeled edges (transitions) and states. The edge presents how many strings traversed the transition, while the state counts a number of strings ended in it. The resulting PA will be created by the iterative merging of states of this prefix tree.

2. From the root state of the prefix tree iteratively search for similar states. For this maintain two sets of states *Red* and *Blue* set. The *Blue* set contains still unprocessed states of the prefix tree. In each iteration take a blue state. If there is a similar red state, merge these two states together (and update values on the transitions starting at the states). The test used to decide whether the states should be merged or not is based on relative frequencies of the empty string and of each prefix. Finally, update the *Red* and the *Blue* set.

3. The automaton constructed in the previous steps contains an integer on each transition. Normalize these values to obtain a PA with probabilities on transitions.

In the description above we first create a prefix tree with number-labelled edges (transitions) and states. The number in a transition expresses how many strings from S traverse this transition in the prefix tree. Similarly, the number at a state represents a number of strings that ends at this state. In the next step, the learning algorithm merges states of

this prefix tree. This leaves us with frequencies, or the number of times an event occurs on each transition and state. To finalize the tree and compute needed probabilities, state and transition frequencies are normalized.

Chapter 3

Industrial Communication IEC 61850

There has always been a need to transfer information. The main objective of a data communication system is to transfer information from one place to another. For many years, people have wanted to standardize a suite of protocols that would meet the common requirements of the electrical utility industry. The last resulting communication standard is known as IEC 61850.

It is defining communication protocols for intelligent electronic devices (IEDs) at electrical substations. Its architecture abstracts the definition of data items and the services by creating data objects and services that are independent of any underlying protocols. The abstract definition allows mapping to any other protocol that can meet the data and service requirements.

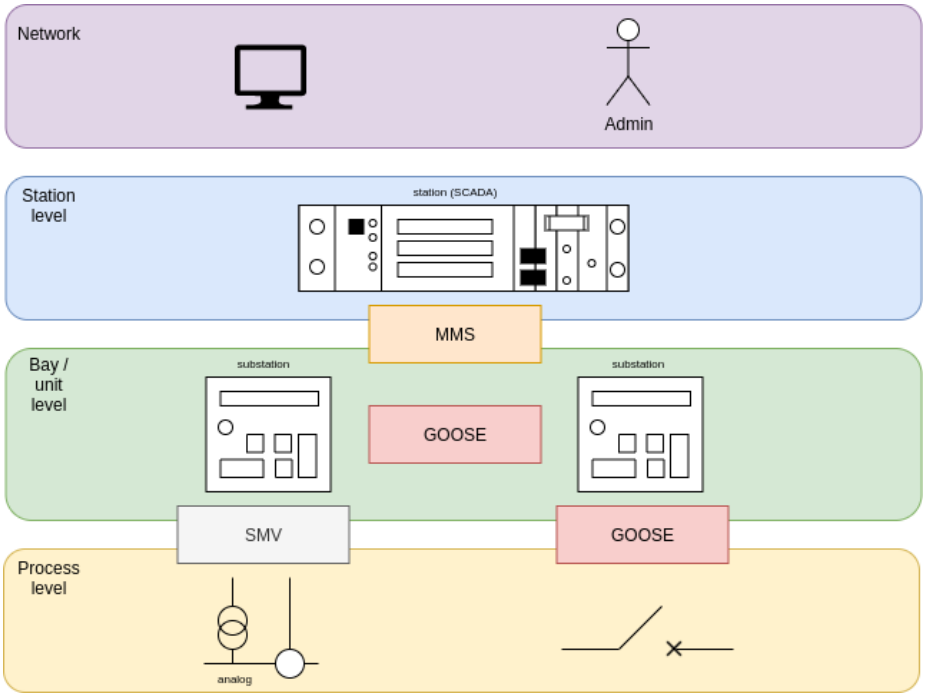

Figure 3.1: IEC 61850 communication scheme

The abstract data models can be mapped to a number of protocols. While IEC 61850 defines three types of communication: MMS (Manufacturing Message Specification), GOOSE (Generic Object Oriented Substation Event), and SMV (Sampled Measured Values), this chapter will focus only on GOOSE and MMS.

Using IEC 61850 standard, measurements can be gathered into a central system. This repository of data can be accessed by multiple users anywhere on the given network.

Key elements of this standard can be described as communication, the definition of semantic data models, and a description of complete engineering processes such as configuration or automation of the network. In this chapter, we will cover mainly communication and data models.

3.1 Communication in IEC 61850

The standard differentiates two main communication approaches, *client-server* and *publisher-subscriber*.

Client-server communication generally consists of request/response sequences. A client requests data or action from the server. Matching request/response sequences and representing them is defined as MMS. In the *publisher-subscriber* [3] method, we publish data to the network. Any device can then subscribe to it. We typically use this method for time-critical information exchange.

It also has a concept where it separates the applications or object models, from the communication. We can describe it as an abstract interface defining from the abstract viewpoint the communication methods. This is then mapped on an existing protocol. We will go into more details about abstract models in part 3.2.

Standard divides the communication infrastructure into three main areas. Between devices within the substation, between devices in the substation and the central control center, and between clients at the control center. As shown in Figure 3.1, IEC 61850 is defining communication on all levels.

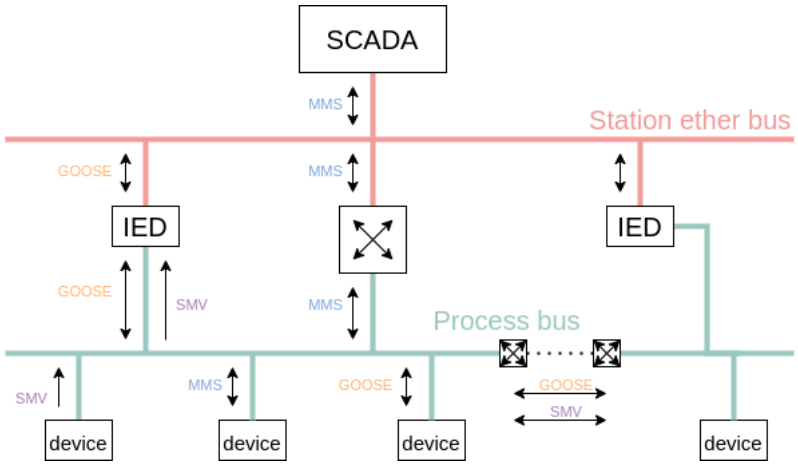

Figure 3.2: Station bus and process bus

For better understanding, we defined communication between station level and bay level as *station bus* and communication between bay level and process level as *process bus*. Although we can find these terms in literature, they are not officially defined in IEC 61850. As shown in Figure 3.2, even if we have the network segregated with routers, potentially

we have all types of traffic everywhere on the network. Station bus and process bus do not divide the traffic. For example, if station control wants to send a control message to the circuit breaker, MMS traffic will go through the whole network. Applying the same principle, GOOSE messages can be found on the station bus.

3.2 Abstract Data Mapping

IEC 61850 supports so-called self-description[9]. This refers to the ability of a device to be browsed to determine what data is contained within it.

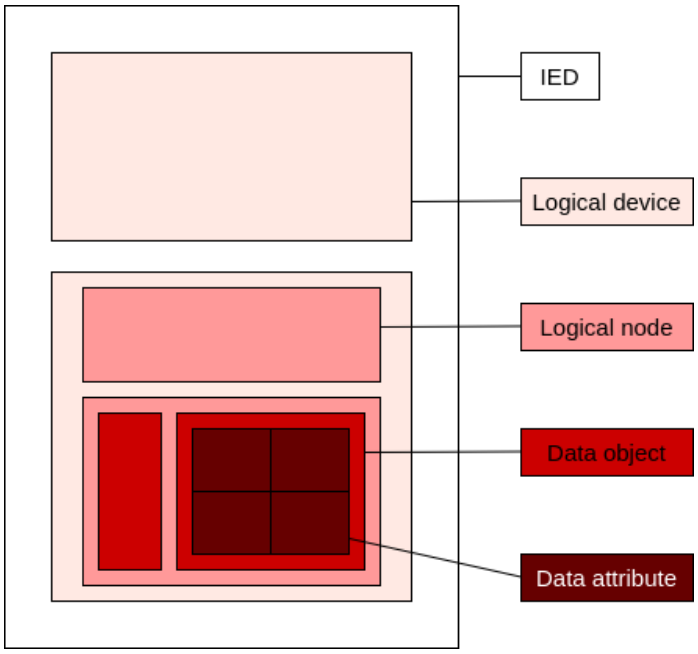

Figure 3.3: Data models in IEC 61850

As shown in Figure 3.3, each *physical device* such as a relay may contain one or more *logical devices*. Each logical device may contain one or more *logical nodes* such as circuit breakers or transformers. Logical node is the main concept of data models representing the grouping of information representing functional elements. It is basically a container for function-related data. Each logical node contains a predefined set of *data classes* that contain *data*.

Once information is described in the location, it is mapped to all locations and never has to be remapped. This allows to reuse the information - once information is modeled in one device, it can be shared with other devices. Because data in logical nodes can be browsed and shared, integration time and cost of applications were reduced with this standard.

Abstract Communication Service Interface

The concept of abstract data mapping allows us to separate the applications (objects) from the communication itself. These objects are accessed via Abstract Communication Service Interface (ACSI).

ACSI describes communication between a client and a remote server. It allows specific mapping as shown in Figure 3.4 and defines the communication methods and configuration

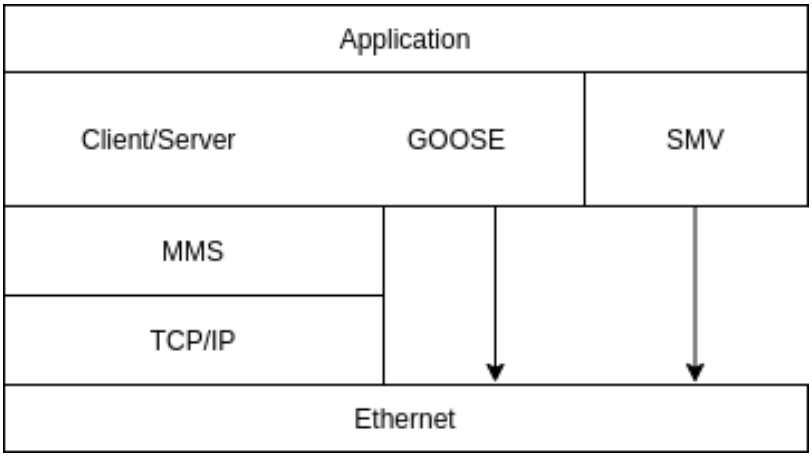

Figure 3.4: Mapping of protocols in IEC 61850

from the abstract viewpoint. This is then mapped on a real communication protocol. Today this is GOOSE and MMS over TCP/IP.

As shown in Figure 3.4, protocols GOOSE and SMV are directly encapsulated in Ethernet. Client/Server is a wider concept of communication and can be mapped to MMS as a transport protocol for client-server services.

3.3 Substation Configuration Language

Given the data and services abstract definitions, the final step is mapping the abstract services into an actual protocol. From a system perspective, there is a significant amount of configuration that is required in order to put all the network nodes together and have them work. In order to facilitate this process and to eliminate much of the human error component, an XML-based Substation Configuration Language (SCL) was defined.

Relation between the substation automation system and the substation is formally described as XML-based Substation Configuration Language (SCL).

3.4 Protocols in IEC 61850

Standard currently defines two types of communication services, *client-server* and *publisher-subscriber*. These services can be mapped to various protocols. Figure 3.5 shows that the client-server uses MMS as its transport protocol. This unicast communication contains commands and reports which are then confirmed. Real-time protocols are time-critical, and they usually use multicast communication. This way packets do not have to be confirmed - for more details about the publisher-subscriber mechanism see part 3.4.1. In standard, they are mapped to GOOSE and SMV.

3.4.1 GOOSE

GOOSE is an Ethernet protocol used for peer-to-peer communication for transmitting IED state information. It is time-critical for the control system to receive the information as soon as possible. That is why GOOSE is characterized by these attributes:

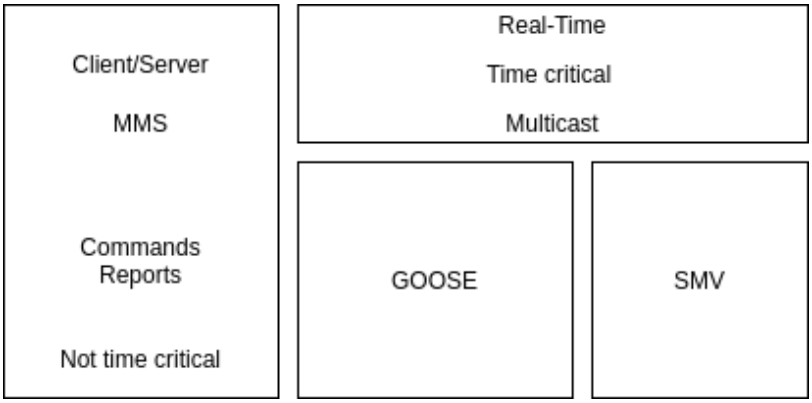

Figure 3.5: Protocols in IEC 61850

- *Unconfirmed transmission* - receiver does not confirm whether the information was received or not. This way the sender does not have to wait for confirmation before sending another packet.
- *Event-driven timing* - data change triggers status updates and is published immediately.
- *Repetition strategy* - packets are repeatedly sent with increasing time for retransmission. This method ensures reliability against packet loss.

This strategy together with the publisher-subscriber mechanism ensures fast and efficient communication.

Publisher-subscriber communication pattern in GOOSE does not send messages directly to subscribers. It instead transmits messages over the Ethernet using a reserved multicast destination MAC address, see Table 3.1. Subscribers can express interest in the data and only receive the information they are interested in without the knowledge of publishers.

Service	Starting MAC address	Ending MAC address
GOOSE	01:0c:cd:01:00:00	01:0c:cd:01:01:ff
GSSE	01:0c:cd:02:00:00	01:0c:cd:02:01:ff
SMV	01:0c:cd:04:00:00	01:0c:cd:04:01:ff

Table 3.1: Recommended multicast address ranges

GOOSE Message Format

As we have shown in Figure 3.4, the GOOSE protocol does not use TCP/IP layers for communication. It is encapsulated directly in the Ethernet frame, see Figure 3.6.

As shown in the figure, four special fields are added in the Ethernet frame following EtherType:

- *AppID* (application identifier) is used as a handle for receiving applications. When not configured, the default value is 0x0000.

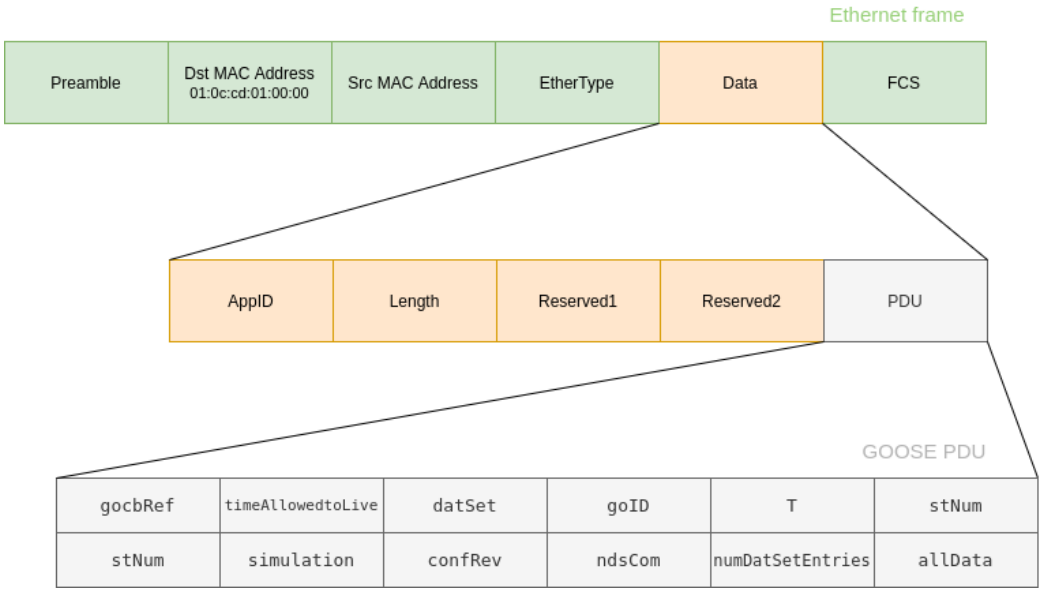

Figure 3.6: GOOSE encapsulated in Ethernet frame

- *Length* is the length of data. Its value starts with AppID and ends at the end of GOOSE PDU.
- *Reserved1* is a 2-byte field. It contains a simulated bit that is mapped from the parameter simulation of the GOOSE PDU, three bits reserved for the future application, and twelve reserved security bits that are used when GOOSE is transmitted with security, otherwise, it shall be set to 0.
- *Reserved2* is also 2-bytes long. All bits are security bits and are used when GOOSE is transmitted with security, otherwise it shall be set to 0.

Structure of GOOSE PDU

Substation configuration language (SCL) files are used to exchange data. On the application layer, GOOSE defined in the SCL file has twelve fields [10]:

```
IECGoosePdu ::= SEQUENCE {
    gocbRef            [0] IMPLICIT VISIBLE-STRING,
    timeAllowedtoLive  [1] IMPLICIT INTEGER,
    datSet             [2] IMPLICIT VISIBLE-STRING,
    goID               [3] IMPLICIT VISIBLE-STRING OPTIONAL,
    T                  [4] IMPLICIT UtcTime,
    stNum              [5] IMPLICIT INTEGER,
    sqNum              [6] IMPLICIT INTEGER,
    simulation         [7] IMPLICIT BOOLEAN DEFAULT FALSE,
    confRev            [8] IMPLICIT INTEGER,
    ndsCom             [9] IMPLICIT BOOLEAN DEFAULT FALSE,
    numDatSetEntries  [10] IMPLICIT INTEGER,
    allData           [11] IMPLICIT SEQUENCE OF Data,
}
```

These fields are defined by IEC 61850 standard [10] as follows:

- *gocbRef* (GOOSE control block reference) is a unique name of an instance of GOOSE control block. The format is *LDName/LLN0.GoCBName*, e.g., *GEDeviceF650/LLN0 GOgcb01*, where LDName is a name of a logical device, logical node class is LLN0 (Logical Node Zero) and GoCBName is the name of the GOOSE control block.
- *timeAllowedtoLive* informs subscribers how long to wait for the retransmission of the packet.
- *datSet* (data set reference) is a reference of the data set whose values will be retransmitted.
- *goID* (GOOSE ID) identifies GOOSE message.
- *T* (event timestamp) is a time at which *stNum* was incremented.
- *stNum* (state number) is a counter that increments each time a GOOSE message has been sent and a value change has been detected within the *datSet*.
- *sqNum* (sequence number) is the current sequence number of the report. It is incremented each time a GOOSE message is sent.
- *simulation* (test bit). If true, the message was sent by the simulation unit and therefore not by the real IED.
- *confRev* (configuration revision) indicates reordering the members of a data, deleting a member, or changing the *datSet* value. The number represents a count of such events.
- *ndsCom* (needs commissioning). If true, GoCB requires further configuration.
- *numDatSetEntries* is a number of data set entries.
- *allData* is a list of user-defined information specified in the control block.

3.4.2 MMS

MMS is an international protocol specified in IEC 61850 standard used for communication and control-order transmissions between a server and a client. It specifies a messaging system for modeling real devices and functions and for exchanging information about the real device[10]. As shown in Figure 3.1, it is mainly used for information exchange between Intelligent Electronic Devices (IEDs) and control systems. Figure 3.4 shows, that MMS is usually addressed by IP addresses, is encapsulated over TCP, and communicates over Ethernet.

This protocol introduces the basic concept of a Virtual Machine Device (VMD). It sets a series of data types on behalf of the real device functions as in Figure 3.3 and hides the specific characteristics of the device through abstract mapping. MMS does not specify the implementation of VMD on real devices.

Through access to the VMD model, one can manipulate how the actual equipment works. The VMD concept of MMS introduces the idea of object-oriented design into the process control system and represents a container in which all other objects are located [10].

MMS is a Client/Server communication protocol. It is used mainly for monitoring, therefore is not time-critical. Communication uses commands which are then confirmed, as in Figure 3.7. In this case, a client (control center) sends atomic operations such as read, write or control commands and requests to the server. The server is typically IED containing VMD and its objects. It responds to the client with data, reports, etc.

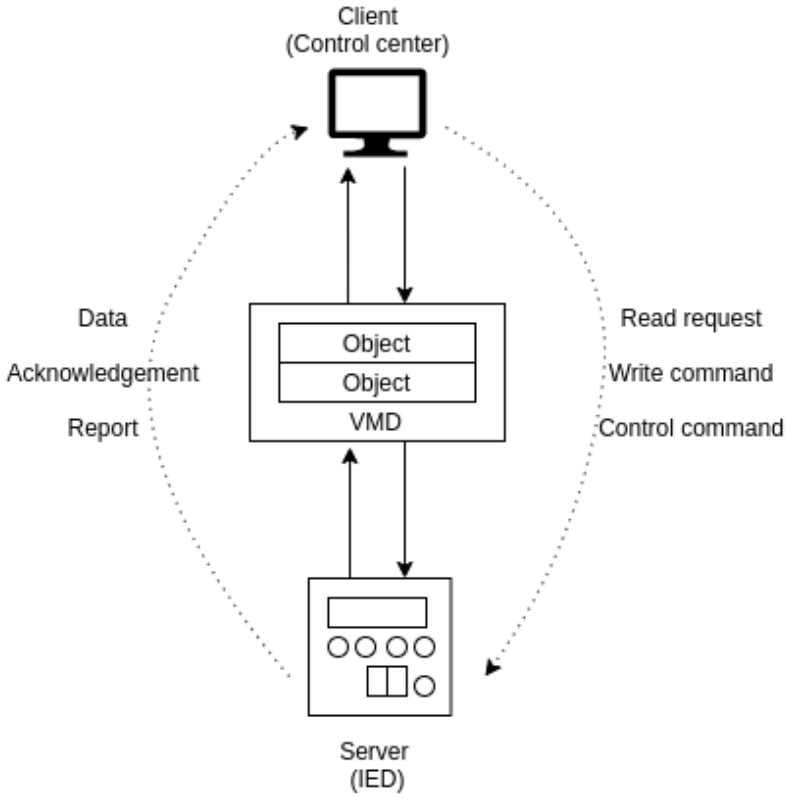

Figure 3.7: Client/Server communication

Virtual Manufacturing Device

VMD model does not specify details about how a real device implements the model. It is focused only on network-related aspects of communication, such as:

- *objects* (variables) and their *attributes* (name, type, etc.) that are contained in the server,
- *services* for accessing objects, such as read, write commands,
- *behavior* that a device should follow when processing the services.

MMS defines a variety of objects. All objects except unnamed variables are identified by an object name. Unnamed variables are identified by a fixed physical address. For each object, the standard defines the corresponding service. The table showing how this mapping is defined is copied from [10] and can be found in Appendix B.

Data Transfer in MMS Protocol

However MMS does not specify an addressing scheme of client-server communication and relies fully on underlying protocols, nodes are usually addressed by their IP address. MMS is encapsulated over TCP port 102. Encapsulation includes several ISO protocols, see Figure 3.8 [10]. The following text is focusing on the L7 layer, as it is the most important one for this thesis. The rest will not be covered.

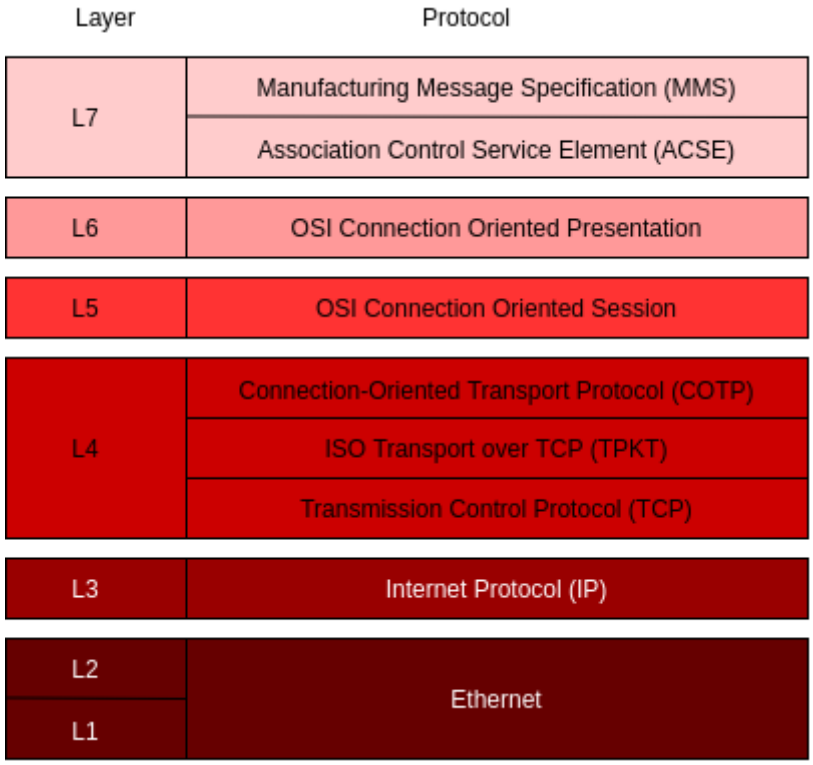

Figure 3.8: ISO/OSI MMS encapsulation over TCP/IP

The MMS standard requires the MMS service to be realized using Association Control Service Element. This is an OSI standardized element of the application layer, that is responsible for communication between network nodes. It establishes, controls, and terminates such connection. MMS packets are only encapsulated in ACSE in the opening phase, not during data transmission.

Communication in MMS

As shown in Table 3.2 from [10], the standard defines 14 types of MMS PDUs. They are identified by TLV identifier with the tag number.

Initiate-Request and *Initiate-Response* open the connection. They are used to negotiate details of the connection and supported services. *Confirmed-Request* is confirmed by *Confirmed-Response* or *Confirmed-Error*. These PDU types create the majority of the communication. They are responsible for transmitting various data based on the type of request. *Confirmed-Request* can be canceled by *Cancel-Response* PDU which is then confirmed by *Cancel-Response* or *Cancel-Error*. Unconfirmed services have PDU type *Unconfirmed* and can be invoked by servers only. They enable IEDs to notify a client when a

MMS PDU	TLV Identifier	Tag Number
Confirmed-Request	0xa0	0
Confirmed-Response	0xa1	1
Confirmed-Error	0xa2	2
Unconfirmed	0xa3	3
Reject	0xa4	4
Cancel-Request	0x85	5
Cancel-Response	0x86	6
Cancel-Error	0xa7	7
Initiate-Request	0xa8	8
Initiate-Response	0xa9	9
Initiate-Error	0xaa	10
Conclude-Request	0x8b	11
Conclude-Response	0x8c	12
Conclude-Error	0xad	13

Table 3.2: Types and identifiers of MMS PDUs

predefined event has occurred. *Conclude-Request* is usually followed by *Conclude-Response* or *Conclude-Error*. This type of message is used to close the connection to the server. This allows to properly release the resources used for the connection. If the server denies the conclude request, the connection remains open.

The following example shows an example of MMS PDU of type *Confirmed-Request* that consists of Type-Length-Value (TLV) encoding scheme.

Example 3.4.1. a0 0f 02 02 06 28 a1 09 a0 03 80 01 09 a1 02 80 00

- 0xa0 refers to *Confirmed-Request* PDU type of size 0x0f (15 bytes).
- 0x02 denotes *InvokeId*. This is an integer matching requests with responses. It starts with the type (0x02), its size is 0x02 (2 bytes) and its value is the next two bytes (06 28).
- 0xa1 refers to the type of MMS service, see Appendix B. Tag 1 denotes *getNameList* service.

Packet Headers Encapsulating MMS

To be able to read desired fields of the PDU, one needs to know how exactly the packets are encapsulated. Figure 3.9 shows, how the PDU of the *Confirmed-Request/Confirmed-Response* message looks like during data transmission. MMS PDU must be less than or equal to 120B. All of the packet headers consist of TLV encoding scheme.

TPKT (ISO Transport over TCP) and COTP (Connection-Oriented Transport Protocol) encapsulate MMS packets on the transport layer. *TPKT*implements ISO Transport Protocol Class 0 (TP0). The difference between the TCP and TP0 is that TCP transports a continuous stream of data without explicit boundaries[10]. TP0 expects the data to be sent in discrete objects, called Network Service Data Units (NSDUs).

COTP defines several types of messages, see Table 3.3 where z means credit field used for flow control. MMS communication mostly uses CR and CC for the establishment of

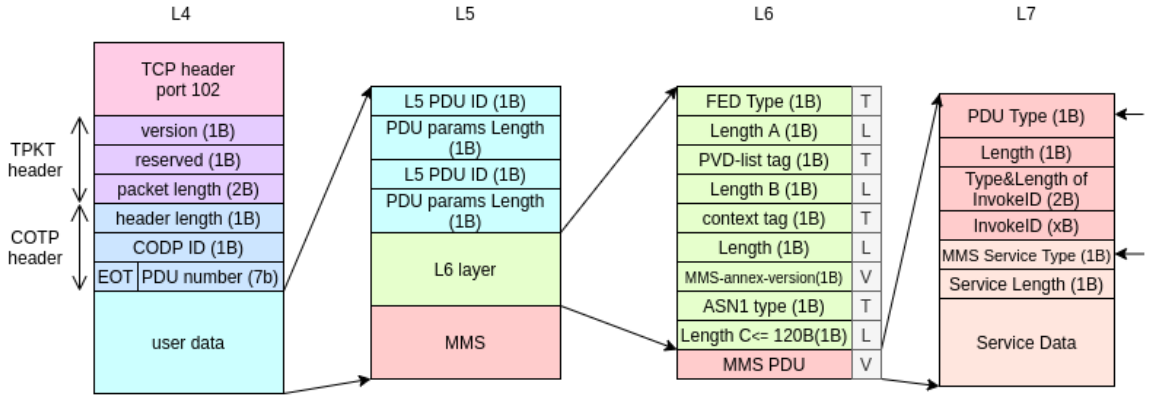

Figure 3.9: Protocols encapsulating MMS packet of type *Confirmed-Request/Confirmed-Response* with MMS PDU lesser than 120B transfering data

connection and DT for data transmission. The structure of the packet varies based on the COTP ID. Messages CR and CC establish transport session and can transmit various parameters such as source and destination reference or maximum size of the PDU. User data are not permitted in these types of messages. COTP DT has a fixed header length (2 bytes). PDU Number is always set to zero for TP0.

Message	COTP ID
Connection Request (CR)	0xDz
Connection Confirm (CC)	0xCz
Disconnect Request (DR)	0x80
Disconnect Confirm (DC)	0xB0
Data (DT)	0xF0
Expedited Data (ED)	0x10
Data Acknowledgement (AK)	0x6z
Expedited Data ACK (EA)	0x20
Reject (RJ)	0x5z
TPDU Error (ER)	0x70

Table 3.3: COTP ID - message types

OSI session protocol on L5 defines connection-oriented session services. While the standard defines more than 30 types of PDUs, MMS employs three of them: Connect (L5 PDU ID = 13), Accept (L5 PDU ID = 14) and Give Tokens, Transfer Data (L5 PDU ID = 1). The last type can be seen in Figure 3.9. After the connection is established by successful exchange of Connect and Accept messages, MMS messages are encapsulated in a sequence of two L5 PDU headers, *Give Tokens* and *Data Transfer*, since the standard does not allow to send *Data Transfer* PDUs individually. Both PDUs are without parameters.

OSI presentation layer L6 provides negotiation during connection establishment as well as data encoding during data transfer. MMS only uses three types of PDUs: Connect-Presentation (CP) type, Connect Presentation Accept (CPA) type, and CPC type, which can be seen in Figure 3.9 and contains user data. The PDU starts with application-specific type which denotes data type *Fully-Encoded-Data* (FED Type). Next, the length of data

and a new TLV structure with PVD-list (Presentation Data Values) tag follows. The next embedded TLV structure starts with a *presentation context tag*. Its value is negotiated in CP and CPA messages. The last TLV structure is a context-specific data type with tag 0 (*ASN1 Type*). This means that the PVD-list contains only one presentation data value which is a single ASN.1 type[10].

The last PDU in Figure 3.9 is the *MMS* PDU with a size smaller than 120 bytes. It is of type *PDU Type*, which can have values as in Table 3.2. The *confirmed-request* PDU has the same *InvokeID* as the *confirmed-response* message with the request result. Next is *MMS Service Type* which shows how objects in VMD are mapped to the corresponding service, see Appendix B.

Chapter 4

Profiling Tool

This chapter will provide details about anomaly detection in GOOSE and MMS protocols and how we implemented them. Various statistical analysis methods and probabilistic automata will be used and the results will be compared.

GOOSE protocol is stable, with simple and unchanging communication patterns. Various statistical methods can be used for detection. The detecting tool was created for the purpose of this thesis, see Appendix A. MMS is also stable, but communication patterns can change. It is important that they do not change very often and are periodical, therefore probabilistic automata can be used. The modified version of the tool `detano` [7] [6] was used for detection, see Appendix 2. In this chapter, a description of the tool created for protocol GOOSE will be provided, as well as a description of the datasets and detection methods used, including probabilistic automata.

In section 4.1, we are going to show the structure of the tool created for detection in protocol GOOSE. Section 4.2 will cover how the data are processed when they are input to the GOOSE tool, as well as how the probabilistic automata will be learned.

4.1 Tool Structure

The profiling tool is divided into two basic components: the learning phase and the detecting phase. Both can be found in `src/Learning.cpp` in function `void learn_and_detect()`. It has three input parameters, namely path to *file to use for learning*, path to *file to use for detecting*, and the statistical detection method used for GOOSE. The tool can use three detection methods, *arithmetic mean and variance*, *box plot*, and *confidence interval*, see Chapter 2.

A detailed description of how to use a profiling tool can be found in Appendix A.

4.1.1 Learning Phase

As was already mentioned in part 4.1, detection for GOOSE protocol can use three statistical analysis approaches, detection for MMS protocol uses probabilistic automata. Before using any method, the tool first needs to divide the file into time slots, or *windows*. The window size can be adjusted in the file `include/Config.h` using constant `WINDOW_SIZE`. The default value used in experiments is 300 seconds or five minutes. If the window is less than 5 minutes, the metrics are incomplete and can corrupt the result. That is the last window is dropped if incomplete.

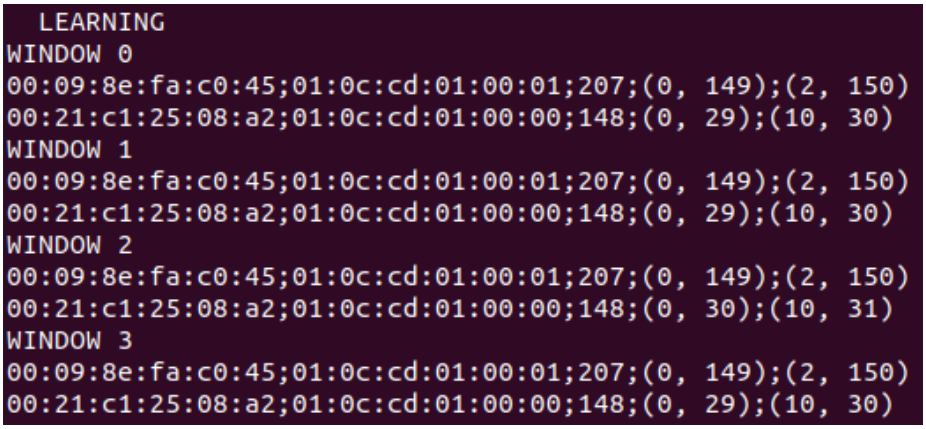

Figure 4.1: Windows and flows of GOOSE communication

As we can see in Figure 4.1, each window is then divided into *flows*. Flow is characterized by unique *source MAC address*, *destination MAC address*, and *length of the packet*. Each flow can then have various *deltas*. Delta is the time between two packets in one flow. It is counted for all pairs of packets that were captured directly in a row. Deltas are stored in tuples together with the total sum of occurrences of the given delta. In Figure 4.1, each flow has two different deltas. For example, in the first flow in `WINDOW 0`, two different deltas were found. The delay between packets of size 0 has occurred 149 times, while the delay of size 2 occurred 150 times.

For the *arithmetic mean and variance* method, see part 2.1.1, the tool computes arithmetic mean and variance of all occurrences of deltas across all windows.

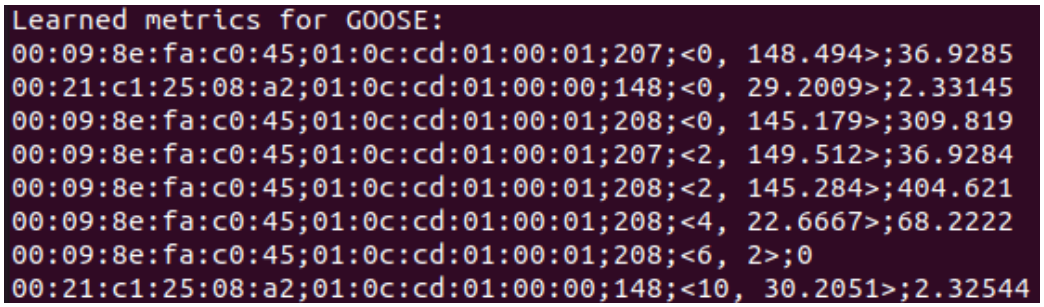

Figure 4.2: Learned GOOSE communication - arithmetic mean and variance

In Figure 4.2 is the learned dataset of GOOSE communication. Data are still divided into flows, according to *source MAC address*, *destination MAC address*, and *length of the packet*. Each flow is then divided based on the value of delta, following by arithmetic mean and variance of delta occurrences. For example, the first flow in Figure 4.2 is of size *207* with delta *0*. The arithmetic mean of deltas that occurred in all flows is *148.494* and its variance is *36.9285*.

For the *box plot* method, see part 2.1.2, five metrics are counted:

- Q2 - a median of the data,
- Q1 - a median of the data on the left of the median,
- Q3 - a median of the data on the right of the median,

- minimum,
- maximum.

Everything before *minimum* and after the *maximum* is considered an outlier.

```
  LEARNING

Learned metrics for GOOSE:
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<0 --> 150;150;150;150;150
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<2 --> 149;149;149;149;149
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<0 --> 28.5;30;30;31;32.5
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<10 --> 27.5;29;29;30;31.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<4 --> -15.5;10;27;27;52.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<6 --> 1;1;1;1;1
```

Figure 4.3: Learned GOOSE communication - box plot metrics

In Figure 4.3 is the dataset learned with the Box Plot learning method. Characteristics of the flow are *source MAC address*, *destination MAC address*, *length of the packet*, and *delta*, following by box plot characteristics *minimum*, *Q1*, *Q2*, *Q3*, and *maximum*. Since GOOSE communication from its nature does not change very often, we can see that the box plot values are very close to each other, or the same.

The third statistical method used is the *confidence interval*. Specifically, the *three-sigma rule* is used, see part 2.1.3. The *three-sigma rule* says that data within three *standard deviations* from the *arithmetic mean* are considered confidence interval. Everything outside this interval is considered an outlier. Because standard deviation can be calculated using variance, a special method for learning the dataset is not necessary, the same learning method as for *arithmetic mean and variance* can be used.

4.1.2 Detecting Phase

For detecting anomalies in GOOSE communication, the same method that was used in learning phase will be used. Functions `detect_anomalies_arithmetic_mean()`, `detect_anomalies_boxplot()` and `detect_anomalies_3_sigma()` are used for GOOSE.

More details about how various statistical analysis methods and probabilistic automata were used can be found below in part 4.2.

4.2 Data Processing

Network traffic in *csv* file format represents input data. Data in this file are learned by the profiling tool line by line. When data are learned, they are divided into chunks called windows. Each window contains 5 minutes of communication. It is possible that the last window will contain a smaller chunk of communication, not the whole 5 minutes. Because of this, the last window is discarded as incomplete.

Data can be further divided into flows representing communication between 2 nodes of the network. Flows are divided by source(*srcMAC*) and destination(*dstMAC*) MAC addresses. Communication attempts in flow are counted, giving us the number of communication attempts in each flow (*count*).

For GOOSE, they can be further divided by *packet length* (this is represented as interval), status number (*stNum*), GOOSE control block reference (*goCBRef*), GOOSE ID (*goID*), and data set (*datSet*). For the purpose of this tool, only *packet length* was used.

Packets that belong to the same flow are captured in time intervals. The time interval between the two packets is *delta*.

Two flows with the same division characteristics can not be in the same window. Each flow in each window represents one collection of statistical data. Statistical methods are further applied to them, as described in the sections below.

4.2.1 Arithmetic Mean

Arithmetic mean is calculated across all flows of all windows with matching division requirements. This means, that if two flows in two different windows have the exact same *srcMAC, dstMAC, length, delta, etc.*, arithmetic mean($\overline{x}$) and variance(s^2) are calculated from the *count* of their *deltas*. This will give us an interval

$$deltacount \in < \overline{x} - s^2, \overline{x} + s^2 > \tag{4.1}$$

of the accepted amount of packets when detecting with minimum $\overline{x} - s^2$ and maximum $\overline{x} + s^2$. The *count* of new flow in the window is simply compared to learned flows. If the new flow matches the division requirements of the learned one, its *count* has to belong to the interval above. Otherwise, an anomaly is detected.

4.2.2 Box Plot

Arithmetic mean is an effective and easy-to-use method but is extremely sensitive to outliers. For the purpose of this thesis, we decided to also use a box plot, as it provides more metrics than the arithmetic mean that can be averaged and compared and is also less sensitive to outliers.

Box plot is a graphical method. It illustrates data using whiskers and quartiles. While whiskers represent minimal and maximal allowed value, quartiles are dividing data into quarters. Most data are within the first and last quartile (Q1 and Q3) and everything below minimum and above the maximum is considered an outlier.

We used these characteristics in the detection tool. After each 5 minutes window of the detection file, box plot metrics are computed and compared to values learned in the learning phase. Depending on how many deltas were detected in one flow of one window, the tool will detect:

1. If the detected value is between Q1 and Q3, nothing happens.
2. If the detected value is not between Q1 and Q3 but is between *min* and *max*, the potential risk is detected.
3. In all other cases, the tool will define communication as an anomaly.

An anomaly is also detected when a new flow is noticed in the detection phase.

4.2.3 Confidence Interval

In the learning phase, after an arithmetic mean($\overline{x}$) and variance(s^2) are calculated, the confidence interval is created. By default, the *three-sigma rule* is applied as follows:

$$delta - count \in < \overline{x} - 3 * s, \overline{x} + 3 * s > \tag{4.2}$$

where s is the square root of *variance* called *standard deviation*.

After detecting a window, all recognized flows have a final count of *deltas* (delays between packets in the same flow), which are then checked. If they do not belong to the confidence interval, an anomaly is detected.

4.2.4 Probabilistic Automata

In this section, we will show how the MMS automaton was learned. For this purpose, `pa_learning.py` from the tool `detano` was used, see Appendix D. For learning normal communication we used dataset `inside_station_normal.pcap`, see the section below.

The dataset `inside_station_normal.pcap` does not contain any packets for the beginning of the conversation or its end (such as *Initiate-Request* or *Conclude-Request*, see Figure 3.2). That is why a new way of dividing conversation into flows needed to be found. We used the *source TCP port* and *destination TCP port* for this purpose. The server port of the conversation will always be **102** because according to standard it indicates MMS communication, but the client port number is not reserved and is a randomly generated number smaller than 65535. In the dataset, it looked like the ports are repeating themselves. We decided to use this fact to help split the conversation. After experiments, the highest accuracy achieved was *0.977* with the number of packets in one conversation equal to 3. The tool split the conversation into two parts. The first part of size 0.33 out of 1 was the learning part. Accuracy was measured using the second part. Figure 4.4 shows the learned automaton.

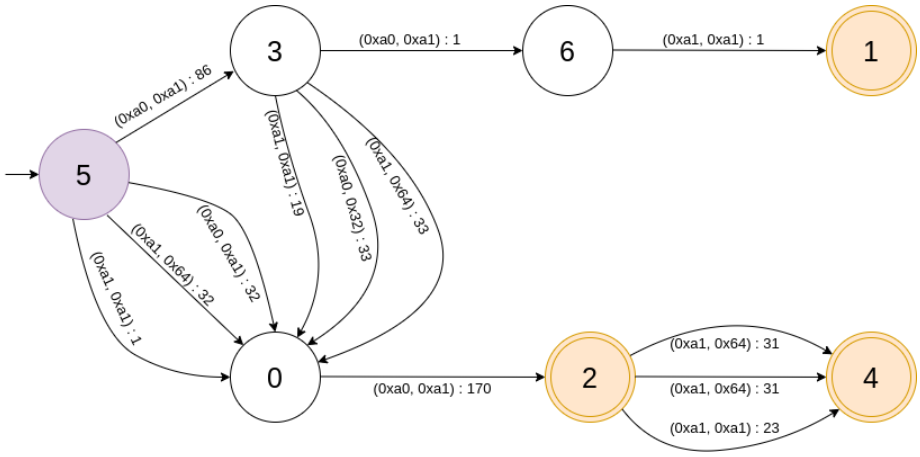

Figure 4.4: Frequency automaton learned from MMS dataset inside_station_normal.pcap

Automata has seven states, marked with numbers. State 5 is the *initial state*. DPA has three *final states* (2, 1, 4). *Transitions* are represented as arrows from one state to another. Their description shows symbols and relative frequencies with which the states were visited through the transition. Another way to represent them is using probability. For example, the communication from state 5 to state 3 used the symbol *(0xa0, 0xa1)* representing tuple *PDU Type, MMS Service*. Its relative frequency to do so was 86 transitions. Frequencies are then normalized to probabilities with which an event occurs.

The learning stage does not divide conversation into windows for learning. In the detecting stage, the conversation was divided into 5-minute windows. After the window

was learned, it was compared to the automaton learned from the normal conversation without anomalies.

4.3 Datasets

The datasets used were provided by [4] and [2]. For the tools, we used the csv (comma-separated values) format of the input files.

gics-goose2.csv

This dataset contains a normal conversation between IEDs in csv format. The fields represent *timestamp*, state number (*stNum*), *source MAC address*, *destination MAC address*, control block reference (*gocbRef*), GOOSE ID (*goID*), data set reference (*datSet*), and *length* of the packet.

To have a better understanding of the data provided, data analysis using programming language Python3 and its libraries `pandas` and `matplotlib` were used. As the name indicates, `matplotlib` is used for creating visualizations and plotting data. `pandas` is primarily written for data manipulation and allows for efficient work with csv files.

`gics-goose2.csv` consists of 83966 packets. Each packet is described by eight variables, as seen above (timestamp, srcMAC, dstMAC, etc.). Variables *timestamp* and *length* are represented numerically, while the rest is represented by a string. Library `pandas` is able to convert string values to integers, allowing us to show a correlation matrix, see Figure 4.5. The matrix shows how variables impact each other. The more two variables depend on each other, the closer correlation is to 1 or -1. Before we computed the matrix, we added variable *deltas* that is the delay between two packets in the same flow.

	timestamp	stnum	srcMAC	dstMAC	gocbRef	goID	datSet	Length	deltas
timestamp	1.000000	NaN	0.001253	-0.001253	-0.001253	-0.001253	-0.001253	0.011852	0.002460
stnum	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
srcMAC	0.001253	NaN	1.000000	-1.000000	-1.000000	-1.000000	-1.000000	-0.999829	-0.157415
dstMAC	-0.001253	NaN	-1.000000	1.000000	1.000000	1.000000	1.000000	0.999829	0.157415
gocbRef	-0.001253	NaN	-1.000000	1.000000	1.000000	1.000000	1.000000	0.999829	0.157415
goID	-0.001253	NaN	-1.000000	1.000000	1.000000	1.000000	1.000000	0.999829	0.157415
datSet	-0.001253	NaN	-1.000000	1.000000	1.000000	1.000000	1.000000	0.999829	0.157415
Length	0.011852	NaN	-0.999829	0.999829	0.999829	0.999829	0.999829	1.000000	0.157424
deltas	0.002460	NaN	-0.157415	0.157415	0.157415	0.157415	0.157415	0.157424	1.000000

Figure 4.5: Correlation matrix of variables in gics-goose.csv including computed deltas between packets

Figure 4.5 shows, that majority of all correlations are close to the absolute value of one, or one. This basically means, that when one variable was changed, the other was also changed. For example, if the source MAC address changed, so did the destination MAC address, length of the packet, etc. The timestamp is not correlated to anything, which is not surprising, because it changes constantly and packets do not have an impact on it. State number is one of the GOOSE-specific variables, see Chapter 3. After further investigation

we found out, that this variable is not changing its value during the communication in the dataset, therefore is not correlated with anything at all.

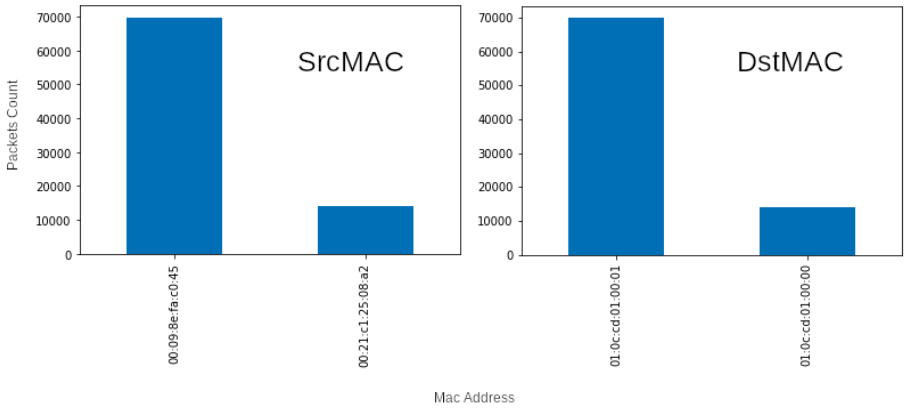

Figure 4.6: Source and destination MAC addresses in gics-goose2.csv

In Figure 4.6 we can see that the source and destination MAC addresses do not overlap. This means, that the conversation was only one-way.

Another important piece of information about the dataset is, that *length* of packets only has three values: 207, 148, and 208. After briefly 60 000 packets when sizes 207 and 148 alternated, the length 207 was changed for value 208, particularly in the window number 167, as will be shown in Chapter 5.

For anomaly detection, we picked variables *source MAC address*, *destination MAC address* as they can separate communication to flows between IEDs, *length*, and *deltas*.

bad_gics-goose2.csv

Since the malformed version of GOOSE communication was not provided, it was created from `gics-goose2.csv` by deleting parts of the conversation and altering values of variable *length*. The result is in Figure 4.7. *Destination MAC address*, *length*, and *deltas* variables were plotted in the 3D plot. After the changes, datasets look visibly different.

inside_station_normal.pcap, mms.pcapng, mms-kocin2.pcapng

The datasets are describing the normal behavior of communication in IEC 61850. We extracted packets using MMS protocol, as this is the protocol we try to detect anomalies in. Because the csv format is needed for the tool, they needed to be parsed. `pcap_to_csv.py` script was created for this purpose using programming language Python3 and library `scapy`, see Appendix C.

As we can see in Figures 3.8 and 3.9, MMS only uses TCP protocol for the transport layer. Above, still on the transport layer, TPKT and COTP protocols are used. There are also protocols used in the Session, Presentation, and Application layer before the actual use of MMS. We will not go into details of any of these protocols, but basic information about how the headers of these protocols look will be needed. It allows us to extract the information from the MMS packet header that we will need to create a DPA. For this PDU Type, Table 3.2 and MMS Services, Appendix B were picked.

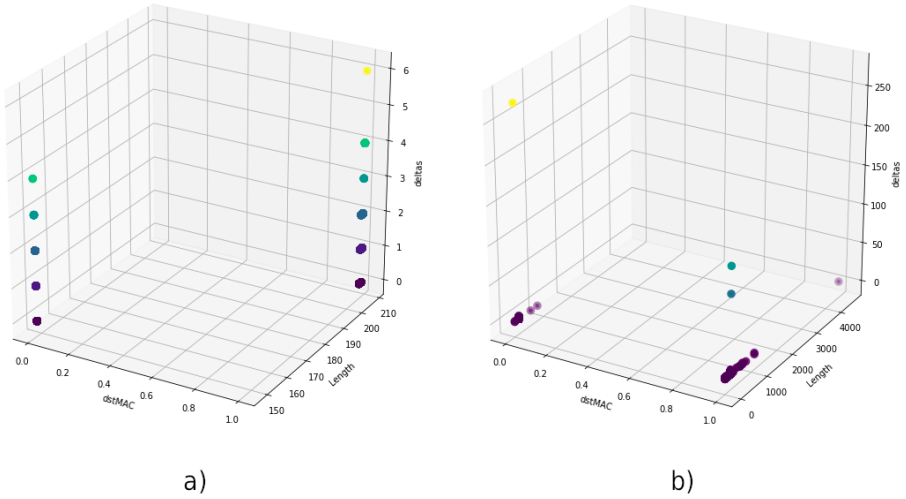

Figure 4.7: Comparison of gics-goose.csv (a) and bad_gics-goose.csv (b)

To be able to read the upper layers, knowledge from Chapter 3.4.2 was used. Everything above the TCP header is considered user data and is not properly recognized in the `scapy` library. Below is an example of this part of a header.

Example 4.3.1. 03 00 00 26 02 F0 80 01 00 01 00 61 19 30 17 02 01 03 A0 12 A0 10 02 03 02 1A 68 A1 09 A0 03 80 01 09 A1 02 80 00

- 0x03 0x00 0x00 0x26 is a TPKT header of version 3 (0x03). Next field is a reserved byte containing 0 and the last two bytes are length of the whole TPKT header including user data, which is 38 bytes.
- 0x02 0xF0 0x80 is a COTP header of length 2 bytes (0x02). This is only header itself without length byte and without user data.
- 0x01 0x00 0x01 0x00 is an L5 header. The firts byte describes PDU ID. It can have three types - 0x0D (Connect), 0x0E (Accept), or 0x01 (Give tokens and transfer data). Next byte shows length of parameters, in this case 0 bytes for *give tokens*. The same for *transfer data*.
- 0x61 0x19 0x30 0x17 0x02 0x01 0x03 0xA0 0x12 describes L6 header of type 0x61, which means, that this PDU transfers data. It is 25 bytes long without the length field (0x19).
- 0xA0 0x10 0x02 0x03 0x02 0x1A 0x68 0xA1 0x09 0xA0 0x03 0x80 0x01 0x09 0xA1 0x02 0x80 0x00 is the actual MMS PDU. It is of the type 0xA0, which means *Confirmed-Request*, see Table 3.2. It has a payload of 16 bytes (0x10). The *Invoke ID* is of type integer (0x02), length 3 bytes and value 137832 (0x02 0x1A 0x68). The MMS service type is 0xA1, *GetNameList-Request*. PDU Type and Service Type will be used in the dataset do differentiate communication flows.

`mms_normal.csv`, `mms.csv`, and `mms-kocin.csv` which were created from these datasets create an abstraction layer over the MMS packet. They contain a *timestamp*, *source IP address*, *destination IP address*, *source TCP port*, *destination TCP port*, *PDU Type*, and *MMS Service*.

bad_mms.pcapng, bad_mms-kocin.pcapng

The malformed datasets were created by altering some of the values in `mms.csv` and `mms-kocin.csv` to simulate new unknown communication patterns and by removing some communication to simulate device outage.

inside_substation_attack3.pcap

This dataset is supplementing version of the communication with anomalies, that need to be found. It is not crucial whether an attack was performed and what type of attack, or whether some communication is missing, rather than the file itself is incomplete and will not contain the same patterns of transmission as the dataset describing normal behavior. Using the script for parsing pcap files to csv files, Appendix C, `mms_anomaly.csv` was created from the dataset.

Chapter 5

Experiments

In this chapter, we are going to experiment with proposed approaches. For GOOSE, we used a detection tool created for this purpose. It was described in chapter 4, see Appendix A. The modified version of the tool `detano` [7] [6] was used for detection in protocol MMS, see Appendix D.

5.1 Protocol GOOSE and Statistical Analysis

In this section, we are going to experiment with the GOOSE protocol and the different approaches of statistical analysis we proposed above. At the end of this section, a conclusion with a comparison of methods will be provided.

5.1.1 Experiment 1 - Arithmetic Mean

The first experiment with detection using the arithmetic mean was to learn and detect communication patterns using the same file. This was mainly to check the correctness of the tool. The expectation was that tool would not find any anomalies. For this purpose dataset `gics-goose.csv` was utilized.

```
File to learn from   : gics-goose2.csv
File for detection   : gics-goose2.csv
Method used for GOOSE: Arithmetic mean

  LEARNING

Learned metrics for GOOSE:
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;207;<0, 148.494>;36.9285
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;<0, 29.2009>;2.33145
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;<0, 145.179>;309.819
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;207;<2, 149.512>;36.9284
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;<2, 145.284>;404.621
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;<4, 22.6667>;68.2222
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;<6, 2>;0
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;<10, 30.2051>;2.32544

  DETECTING

WINDOW 167
 -> ANOMALY! delta count: 71 min allowed: 111.566 max allowed: 185.423
   delta size: 0 src: 00:09:8e:fa:c0:45 dst: 01:0c:cd:01:00:01 size: 207
WINDOW 167
 -> ANOMALY! delta count: 70 min allowed: 112.583 max allowed: 186.44
   delta size: 2 src: 00:09:8e:fa:c0:45 dst: 01:0c:cd:01:00:01 size: 207
```

Figure 5.1: Detection using arithmetic mean

As we can see in Figure 5.1, this was not the case. When we take a closer look at the dataset, we can see a change in communication. Specifically, in window 167 seen in Figure 5.2, the packet flow changed its packet size from 207 to 208. Before this window, only packets with size 207 were captured, after this window there were none. The implemented mechanism needed to be adjusted to use the dataset by rounding the packet size.

```
WINDOW 165
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;207;(0, 149);(2, 150)
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;(0, 29);(10, 30)
WINDOW 166
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;207;(0, 149);(2, 150)
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;(0, 29);(10, 30)
WINDOW 167
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;207;(0, 70);(2, 71)
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;(0, 30);(10, 31)
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;(0, 78);(2, 79)
WINDOW 168
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;(0, 149);(2, 150)
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;(0, 29);(10, 30)
WINDOW 169
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;208;(0, 149);(2, 150)
00:21:c1:25:08:a2;01:0c:cd:01:00:00;148;(0, 29);(10, 30)
```

Figure 5.2: Window 167

Rounding the Packet Size

With this modification, we were able to eliminate all of the false positives found before. This feature might not be always needed and that is the reason we introduced configurable constant `ROUND_PACKET_SIZE` to the tool. It is also not clear yet whether this approach will hide anomalies or not.

After the modification of packet size, no other anomalies were found in the dataset when using the same one for both detecting and learning.

Detecting Anomalies in Corrupted File

For this experiment `gics-goose2.csv` file was used for the learning phase. For the detecting phase, we altered some of the values of packet size to create new, unlearned flows. Also, some communication was removed to simulate the outage of an electronic device. This new dataset is saved in the file `bad_gics-goose2.csv`.

The experiment was executed both with and without rounding the packet size. Apart from the anomaly detected in window 167 when the size was not modified, the output was the same. This means that rounding the size of packets was not crucial for classifying communication patterns.

This approach was able to correctly identify all of the new packet sizes. It also found windows where the outage of IEDs was simulated. Together, 69 anomalies excluding false positives caused by window 167 were found. However, changing packet size while creating `bad_gics-goose2.csv` caused another anomaly. Modified packets were subtracted from existing flows causing the flows to have smaller counts of packets in several windows. This change was not detected by the proposed method.

5.1.2 Experiment 2 - Box Plot

As the first experiment with the box plot, we decided to both learn and detect from the same file (`gics-goose2.csv` was used). After the learning phase (details can be found in part 4.1.1) box plot metrics were needed. Our first approach was to count one *Q1, Q2, Q3, minimum and maximum* from the whole detecting file. As figure 5.3 shows, no anomaly was found. This was an expected result. Of course, this approach is not sufficient, because the tool expected multiple windows for detecting at once.

```
File to learn from   : gics-goose2.csv
File for detection   : gics-goose2.csv
Method used for GOOSE: Box Plot

  LEARNING

Learned metrics for GOOSE:
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<0 --> 150;150;150;150;150
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<2 --> 149;149;149;149;149
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<0 --> 28.5;30;30;31;32.5
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<10 --> 27.5;29;29;30;31.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<4 --> -15.5;10;27;27;52.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<6 --> 1;1;1;1;1

  DETECTING

OK
```

Figure 5.3: Detection using box plot

Our next approach was to learn one box plot for each flow using all windows from the learning file. The goal was to compare these learned metrics with each window from the detecting file separately. The expected result was again no errors. Figure 5.4 shows the real result of this approach.

```
  DETECTING

WINDOW 52 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0  Q1: 150 Q2: 150 Q3: 150  min: 150 max: 150
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0 count: 149
WINDOW 52 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 2  Q1: 149 Q2: 149 Q3: 149  min: 149 max: 149
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 2 count: 148
WINDOW 102 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0  Q1: 150 Q2: 150 Q3: 150  min: 150 max: 150
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0 count: 149
WINDOW 102 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 2  Q1: 149 Q2: 149 Q3: 149  min: 149 max: 149
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 2 count: 148
WINDOW 108 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0  Q1: 150 Q2: 150 Q3: 150  min: 150 max: 150
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 0 count: 149
```

Figure 5.4: Detection using box plot, comparing each detected window independently

As we can see in Figure 5.4, this approach showed multiple false positives. This is because of how the pox plot metrics are counted. After sorting all of the deltas in one flow in ascending order, Q1, Q2, and Q3 will be medians of different groups of data, see 2.1.2. When enough data have the same value, Q1 and Q3 can be the same number. This results in problems when counting the interquartile range(IQR). When IQR is zero, the *minimum* and *maximum* of the boxplot are equal to Q1 and Q3. Any value different than these metrics is considered an outlier. As shown in Figure 5.4 in the beginning, according

to learned values, the delta count in every flow should be 150. But in window 52, only 149 deltas were found. This is classified as an anomaly.

Again we met with the problem of packet size (window 167, where the size of a packet is changed from 207 to 208). Rounding the packet size helped us to get rid of this problem, but there were plenty more false positives. We encountered 17 errors. That is why we decided to try another approach.

```
File to learn from   : gics-goose2.csv
File for detection   : gics-goose2.csv
Method used for GOOSE: Box Plot

  LEARNING

Learned metrics for GOOSE:
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<0 --> 150;150;150;150;150
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<2 --> 149;149;149;149;149
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<0 --> 28.5;30;30;31;32.5
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;<10 --> 27.5;29;29;30;31.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<4 --> -15.5;10;27;27;52.5
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;<6 --> 1;1;1;1;1

  DETECTING

WINDOW 232 potential risk
     -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210 delta: 4 count: 28
OK
```

Figure 5.5: Detection using box plot, learning communication while detecting

In this experiment, the detection tool is also learning communication. Detected windows are not thrown away, rather than stored for further use. New box plot metrics are then computed from the current window, as well as from all previous ones. As shown in figure 5.5, we were able to get rid of almost all false positives. But this approach is only experimental and is questionable how well it can find anomalies.

Detecting Anomalies in Corrupted File

For this experiment `bad_gics-goose2.csv` file was used. We altered some of the values of packet size to create new, unlearned flows in the original `gics-goose2.csv`. Also, some communication was removed. This way we simulated the outage of an electronic device.

In the approach where we learned detected communication, once an anomaly was detected, the tool tried to warn us in all of the following windows, because the anomaly was never forgotten. This experiment showed only a few false negatives, but countless false positives and therefore is not suitable for us.

```
WINDOW 129 potential risk
     -> srcMAC: 00:21:c1:25:08:a2 dstMAC: 01:0c:cd:01:00:00 size: 150
 delta: 10 count: 28
WINDOW 129 NOT OK, unknown flow detected
     Detected -> srcMAC: 00:21:c1:25:08:a2 dstMAC: 01:0c:cd:01:00:00 size: 150
 delta: 20 count: 1
WINDOW 129 packet count does not fit
     Learned  -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 0  Q1: 150 Q2: 150 Q3: 150  min: 150 max: 150
     Detected -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 0 count: 143
```

Figure 5.6: Detection using box plot in corrupted file

Several types of errors were captured, see Figure 5.6. *Potential risk* means, that although delta count is between *min* and *max* of boxplot, it is not between *Q1* and *Q2* and so there

is a risk of corruption. This is not a real error, rather than a warning that can be used to further analyse the flow. *Unknown flow detected* shows that there were characteristics of the flow that were never captured before. *Packet count does not fit* indicates, that delta count was not between *min* and *max* of a boxplot. In this experiment, 90 anomalies were detected. All of the created inconsistencies were revealed.

This approach seems to be more sensitive to small changes in communication than anomaly detection in part 5.1.1. It found 11 more anomalies. All of them were valid. They were created because some of the packets were changed. This left rest of the packets in a window with a smaller count. This change was detected by box plot, but not by arithmetic mean method.

5.1.3 Experiment 3 - Confidence Interval

The first experiment with detection using confidence interval was to learn and detect communication patterns from the same file, to check the correctness of the tool, and to find false positives. The expectation was that tool would not find any anomalies. We are only experimenting with packet size rounded.

```
Learned metrics for GOOSE:
source MAC;destination MAC;packet size;delta;arithmetic mean;variance
srcMAC;dstMAC;0;0;0;0
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;0;149.201;68.9297
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;0;30.2009;2.33145
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;2;147.94;99.5691
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;4;21.6667;68.2222
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;6;1;0
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;10;29.2051;2.32544

  DETECTING

WINDOW 231 anomaly!
     -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 0 count: 120
WINDOW 231 anomaly!
     -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 2 count: 90
WINDOW 232 anomaly!
     -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 0 count: 123
WINDOW 232 anomaly!
     -> srcMAC: 00:09:8e:fa:c0:45 dstMAC: 01:0c:cd:01:00:01 size: 210
 delta: 2 count: 95
actual window: 233
```

Figure 5.7: Detection using confidence interval of 3 sigma

As shown in Figure 5.7, anomalies were found. The dataset consists of 233 windows. The last window is ignored because it is incomplete. This means that in the last two complete windows anomaly was found. When comparing learned metrics one can find, that about 150 deltas of the size of packet 210 were expected, but only 120 found. If we take look at the windows in Figure 5.8 this is really the case. Since we are now learning and detecting from the same file, this is considered a false positive.

```
WINDOW 231
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;0;120;2;90;4;28;6;1
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;0;30;10;29
WINDOW 232
00:09:8e:fa:c0:45;01:0c:cd:01:00:01;210;0;123;2;95;4;27
00:21:c1:25:08:a2;01:0c:cd:01:00:00;150;0;30;10;29
```

Figure 5.8: Windows 231 and 232 of gics-goose2.csv

Detecting Anomalies in Corrupted File

For this experiment, `gics-goose2.csv` was used for learning and `bad_gics-goose2.csv` was used to detect from. This method found 53 anomalies from which 4 were false positives, as in Figure 5.8.

5.1.4 Conclusion and Evaluation of the Statistical Methods

In this section, we will be comparing the results of different statistical methods when used for anomaly detection for the GOOSE protocol. We are only comparing the best results we were able to achieve with each method. That for example means that packet size was rounded in all experiments. When evaluating the box plot method, the type of error message *potential risk* was not taken into account, as this is not an error, rather than a warning that can be used for further experimenting.

The disadvantage of this approach is that since we are looking at the delta between packets, at least two packets with the same values (MAC addresses and length) need to be captured in one window in order to recognize them. From the nature of the GOOSE protocol where the device is sending the same packet multiple times in a row, we assume that this will not cause a problem.

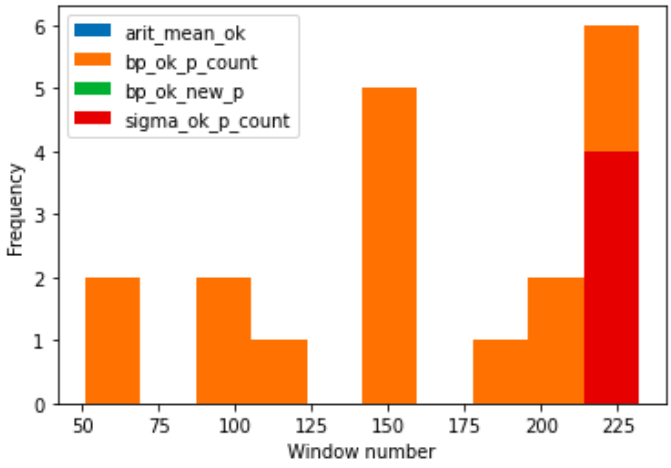

Figure 5.9: False positives detected by methods when detecting from gics-goose2.csv

Figure 5.9 shows how many errors the methods found when learning and detecting from the same file (`gics-goose2.csv`). All of these are classified as *false positives*. *Arithmetic mean* did not find any errors, while both *box plot* method and *3 sigma rule* did. As we can see, the box plot found the most errors. The reason for this is explained in section 5.1.2,

under the box plot method. No new flows were found, all the errors were due to bad packet count.

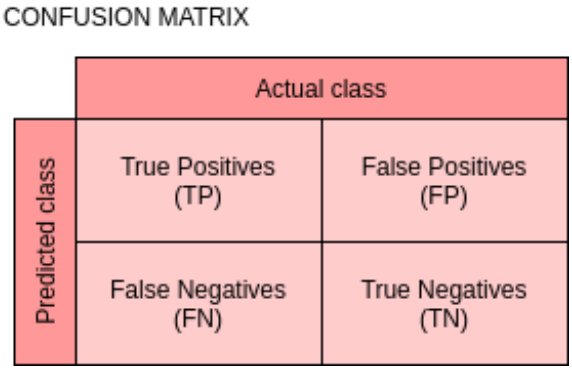

Figure 5.10: Confusion matrix

For measuring the performance of the methods, we chose to use the *Confusion matrix*[5], also called the error matrix, see Figure 5.10. It allows for visualization of error rate and provides variables that can help to calculate useful measurements, such as *classification accuracy*, *error rate*, *sensitivity*, *precision*, etc., where

$$accuracy\_score = \frac{TP + TN}{TP + FP + TN + FN}, \tag{5.1}$$

$$error\_rate = \frac{FP + FN}{TP + FP + TN + FN}, \tag{5.2}$$

and

$$sensitivity = \frac{TP}{TP + FN}. \tag{5.3}$$

Table 5.1 shows how many TP, TN, FP, and FN were found when learning from `gics-goose2.csv` and detecting from `bad_gics-goose2.csv`, as well as their accuracy, error rate, and sensitivity. It is visible from the table that the arithmetic mean used as a statistical analysis tool has the highest accuracy and sensitivity score and the lowest error rate of the tested methods.

5.2 Protocol MMS and Deterministic Probabilistic Automata

In this section, we are going to experiment with protocol MMS and DPA. At the end of this section, a conclusion about this method will be provided.

When creating a detection tool, communication needs to be divided into flows. One flow represents one conversation between devices on the network. When analysing the datasets provided, we found out that they did not contain *Cancel*, *Initiate* or *Conclude* types of MMS messages, see Table 3.2. Because of this, we could not divide the conversation according to the beginning and the end. That is why we decided to split flows according to packet

	Aritmetic mean	Boxplot	3Sigma
True Positive (TP)	69	69	53
True Negative (TN)	859	843	858
False Positive (FP)	0	12	2
False Negative (FN)	8	12	23
accuracy score	0,99145	0,97435	0,97018
error rate	0,0085	0,0256	0,0267
sensitivity	0,8961	0,8518	0,6973

Table 5.1: Accuracy score of statistical methods

count. For example, when we decide on delimiter 16, every 16 packets will be the beginning of a new conversation, considering also dividing it according to source and destination IP addresses and TCP ports. We experimented with this delimiter and found out that 3 packets in one conversation give the best results. Accuracy was measured by splitting the input file into 2 parts, learning from the first one and detecting from the second part. The ratio used was 1:2 learning and detecting.

It was also essential to find the appropriate abstraction of the packets. MMS header and headers encapsulating it contain various information that could be used to represent the communication. If all of this information was used, it could lead to overlearning of the created DPA. For the purpose of this thesis, we used *MMS Service* and *PDU Type*, see Chapter 3, but other fields of headers could be used as well.

5.2.1 Experiment 1 - mms.csv, bad_mms.csv

Dataset `mms.csv` is our smallest dataset, containing 504 packets of MMS communication between two nodes, containing only one flow divided by IP addresses and ports. The first automata in Figure 5.11 shows this communication. For example, we can get from state 1 to state 3 using transition (0xa0, 0xa1) with frequency 5. The type of message in this transition is *Confirmed-Request* (0xa0) and the MMS service used is *getNameList* (0xa1).

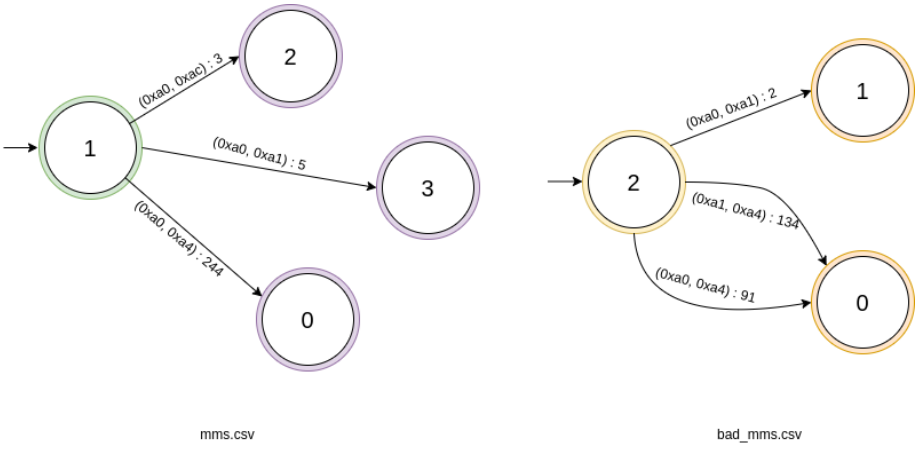

Figure 5.11: Communication learned from mms.csv and bad_mms.csv as frequency automats

When learning and detecting from `mms.csv`, detano [7] tool found out that there are only two nodes, 10.10.20.6 using port 102 (server), and 10.10.3.4 with port 56808 (client). No anomalies were found.

In the case when `mms.csv` was used for learning and `bad_mms.csv` for detecting, the tool found 70 anomalies.

5.2.2 Experiment 2 - mms-kocin.csv, bad_mms-kocin.csv

Dataset `mms-kocin.csv` is the biggest dataset containing 22 529 packets. It consists of 47 different flows divided by IP addresses and ports used. In Figure 5.12 is its learned frequency automaton alongside the automaton learned from `bad_mms-kocin.csv`, which was created altering or deleting some of the values of the original `mms-kocin.csv`.

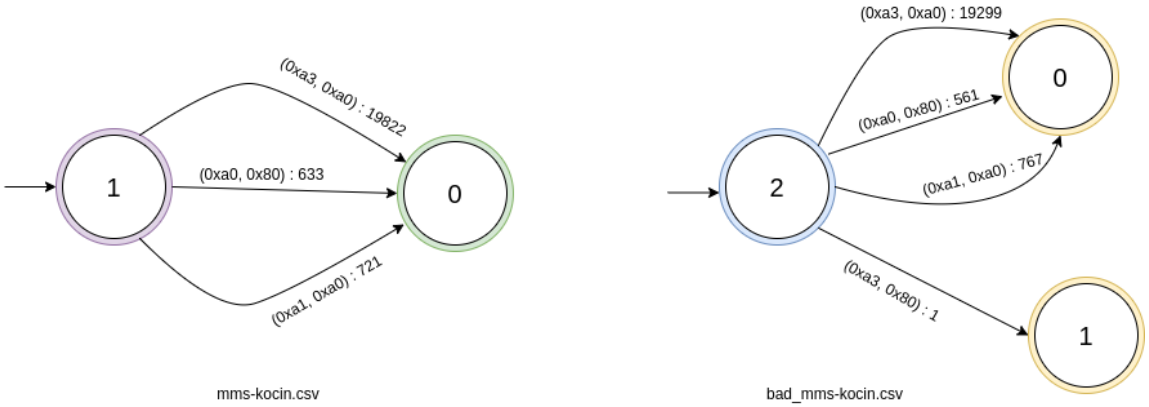

Figure 5.12: Communication learned from mms-kocin.csv and bad_mms-kocin.csv as frequency automats

When learning and detecting from `mms-kocin.csv`, detano tool did not find any anomalies. When detecting from `bad_mms-kocin.csv`, the tool found 10 anomalies.

5.2.3 Experiment 3 - mms_normal.csv, mms_anomaly.csv

File `mms_normal.csv` has 1557 packets of MMS communication divided by detano[7] into 5 flows, see Figure 5.13.

```
frozenset({('192.168.1.52', '102'), ('192.168.1.11', '49623')})
frozenset({('192.168.1.52', '102'), ('192.168.1.12', '35091')})
frozenset({('192.168.1.61', '102'), ('192.168.1.11', '49621')})
frozenset({('192.168.1.11', '49622'), ('192.168.1.51', '102')})
frozenset({('192.168.1.12', '35347'), ('192.168.1.51', '102')})
```

Figure 5.13: Communication flows learned from mms_normal.csv divided by IP addresses and ports

Figure 5.14 shows frequency automats created from `mms_normal.csv` and `mms_anomaly.csv`. The probabilities of the transitions can be counted using the frequencies.

The tool did not find any anomalies detecting from `mms_normal.csv`, but this time, it also did not find any anomalies when detecting from `mms_anomaly.csv`.

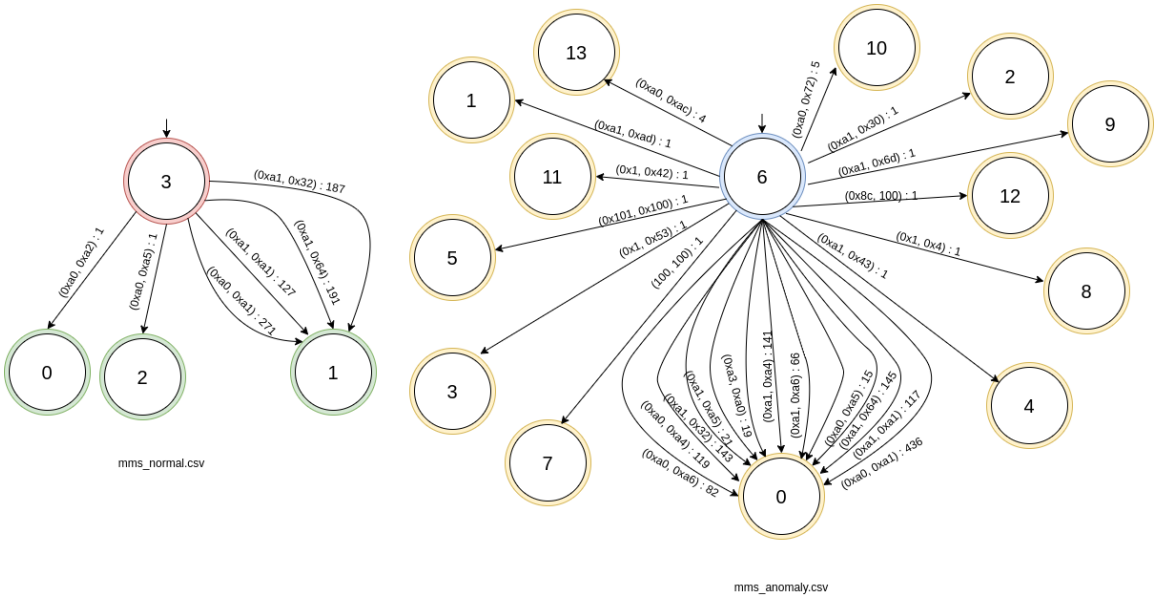

Figure 5.14: Communication learned from mms_normal.csv and mms_anomaly.csv as frequency automats

5.2.4 Conclusion and Evaluation of the Method

Because of the complex nature of the MMS protocol, we are not able to determine how many anomalies that we detected were created in corrupted files and therefore specify concrete values of True Positives, True Negatives, False Positives, and False Negatives.

The results show, that despite incomplete conversations lacking beginnings and ends, DPA works fairly well when dividing communication into groups with the same amount of packets. We were able to correctly find anomalies in corrupted files two out of three times, and no anomalies were found in the files where there were not any. A conversation where we are able to detect *Initiate* and *Conclude* types of messages could help to improve these results. The disadvantage is that the tool found anomalies in files fabricated for the purpose of this thesis, whereas did not detect a real anomaly in the dataset `mms_anomaly.csv`.

Chapter 6

Conclusion

Several vulnerabilities have been exploited in industrial communication in recent years. Many attacks became unnoticed by Intrusion Detection/Prevention Systems and security devices protecting these networks, demonstrating the need to improve their security.

In this thesis, we focused on communication standard IEC 61850 and its protocols MMS and GOOSE. We demonstrated the approach of creating the description of normal communication for GOOSE using statistical analysis. We proposed three approaches. The first one was computing *arithmetic mean and variance* and creating quantity interval within which packets with the same communication patterns were captured. If the interval was not met, the anomaly was detected. The second approach was to create a *box plot* and use its quartiles as an interval. Similarly, *confidence interval* using the *three-sigma rule* was the last approach. We found out that the *arithmetic mean* has the highest *accuracy score* and *sensitivity*, and the lowest *error rate* among the methods tested.

Communication using MMS protocol is not as straightforward as GOOSE, therefore statistical analysis was not used. Instead, we created a communication profile using *deterministic probabilistic automata* (DPA). Transitions between the states were marked with probabilities they would be used with. Algorithm *Alergia* was used for learning and detection. We found out that this approach is successful only for some of the datasets provided and that it was able to find anomalies in 2 out of 3 corrupted files. However, it never found anomalies in the files where there were none.

Further work can be focused on finding appropriate abstraction over the MMS protocol. For this purpose, we used *MMS Service* and *PDU Type*, but other MMS fields could work better for DPA. Decision Trees and Random Forests could help find such fields. It could also be a more relevant detection method. Because of the lack of available datasets of IEC communication, the accuracy of proposed methods is not guaranteed. Samples of the IEC 61850 conversation would be of help.

Bibliography

[1] Dunham, M. H. *Data Mining: Introductory and Advanced Topics*. New Jersey: Prentice-Hall, 2003. ISBN 978-0-13-088892-1.

[2] ENCS. *European Network for Cyber Security* [online]. [cit. 2021-05-11]. Available at: https://encs.eu/.

[3] Eugster, P. T., Felber, P. A., Guerraoui, R. and Kermarrec, A.-M. The many faces of publish/subscribe. *ACM computing surveys (CSUR)*. ACM New York, NY, USA. 2003, vol. 35, no. 2, p. 114–131.

[4] GIPSA lab. *CNRS research unit* [online]. [cit. 2021-05-11]. Available at: http://www.gipsa-lab.grenoble-inp.fr/en/.

[5] Han, J. and Kamber, M. *Data Mining: Concepts and Techniques*. San Francisco: Diane Cerra, 2006. ISBN 9978-1-55860-901-3.

[6] Havlena, V., Holík, L. and Matoušek, P. *Learning Probabilistic Automata in the Context of IEC 104*. 2020. 22 p. Available at: https://www.fit.vut.cz/research/publication/12355.

[7] Havlena, V. *Automata-based Detection of Network Anomalies* [online]. [cit. 2021-05-05]. Available at: https://github.com/vhavlena/detano.

[8] Higuera, C. de la. *Grammatical Inference: Learning Automata and Grammars*. Cambridge University Press, 2010. ISBN 978-1-13-948668-2. Available at: http://pagesperso.lina.univ-nantes.fr/~cdlh/book/allthebook.pdf.

[9] Mackiewicz, R. Overview of IEC 61850 and Benefits. In: *2006 IEEE PES Power Systems Conference and Exposition*. 2006, p. 623–630. DOI: 10.1109/PSCE.2006.296392.

[10] Matoušek, P. *Description of IEC 61850 Communication*. 2018. 88 p. Available at: https://www.fit.vut.cz/research/publication/11832.

Appendix A

Tool manual

Usage

```
$ make
$ ./build/apps/detector learning_file.csv detecting_file.csv -method

-method is method used for detecting anomalies in GOOSE protocol:
  -a : arithmetic mean and variance
  -b : box plot
  -s : confidence interval using 3 sigma rule
```

Input File

File has to be in .csv format.

Configuration

Configuration is needed before the tool can be used. Go to *include/Config.h* and set:

- **INPUT_SEPARATOR** - this should be symbol used to divide fields in your .csv file.
- For GOOSE protocol, set constants position in **goose_layout** (positioning is from 0, not from 1) - *for example, if source MAC address in your input file is on first position, set* goose_layout_SrcMAC *to 0.*

  Required fields:

  - time when packet arrived,
  - source MAC address,
  - destination MAC address
  - packet length.
- **WINDOW_SIZE** - this is size of one window in learning module in **seconds**.
- **DELTA_ROUND_DEC_PLACES** - if you want to have delta in intervals rather than strictly in miliseconds, choose how much your delta shoud be rounded in output.

Appendix B

Mapping IEC 61850 objects and services to MMS

The following table is copied from [10] and shows how ASCI objects and services are mapped to MMS services. It also shows MMS value of the service used in MMS PDU.

IEC 61850 Object	IEC 61850 Services	MMS Object	MMS Services	MMS Value
Server	GetServerDirectory	Virtual Manufacturing Device (VMD)	File Directory	4
Association	Associate		Initiate	1
	Abort		Abort	2
	Release		Conclude	3
Logical Device	GetLogicalDeviceDirectory	Domain	GetNameList	5
Logical Node	GetLogicalNodeDirectory	Named Variable	GetNameList	55
	GetAllDataValues		Read	6
Data	GetDataValues	Named Variable	Read	7
	SetDataVariables		Write	8
	GetDataDirectory		GetVariableAccessAttributes	9
	GetDataDefinition		GetVariableAccessAttributes	10
Data Set	GetDataSetValues	NamedVariableList	Read	11
	SetDateSetValues		Write	12
	CreateDataSet		DefineNamedVariableList	13
	DeleteDataSet		DefineNamedVariableList	14
	GetDataSetDirectory		GetNameVariableListAttributes	15
Setting-Group -Control-Block	SelectActiveSG	Named Variable	Read	16
	SelectEditSG		Read	17
	SetEditSGValue		Write	18
	ConfirmEditSGValues			19
	GetEditSGValue		Read	20
	GetSGCBValues		Read	21
Report-Control-Block	Report	Named Variable	InformationReport	22
	GetBRCBValues		Read	23
	SetBRCBValues		Write	24
	GetURCBValues		Read	25
	SetURCBValues		Write	26
Log		Journal		
Log-Control-Block	GetLCBValues	Named Variable	Read	27
	SetLCBValues		Write	28
	QueryLogByTime		ReadJournal	29
	QueryLogAfter		ReadJournal	30
	GetLogStatusValues		GetJournalStatus	31
GOOSE-Control-Block	GetGoCBValues	Named Variable	Read	33
	SetGoCBValues		Write	34
	SendGOOSEMessage		Write	32
	GetGoReference			35
	GetGOOSEElementNumber			36
GSSE-Control-Block	GetGsCBValue	Named Variable		
	SetGsCBValue			
Control	Select	Named Variable	Read	43
	SelectWithValue		Write	44
	Cancel		Write	45
	Operate		Write	46
	CommandTermination		InformationReport	47
	TimeActivatedOperate		InformationReport	48
Files	GetFile	Files	FileOpen/FileRead/FileClose	49
	SetFile		ObtainFile	50
	DeleteFile		FileDelete	51
	GetFileAttributeValues		FileDirectory	52

Table B.1: MMS Objects and Services

Appendix C

Tools and scripts created for purpose of this thesis

pcap_to_csv.py

This is a script that parses `pcap` file on the input and creates a `csv` file with only a few information found in the PDU headers. In our thesis, we used *timestamp*, *source IP address*, *destination IP address*, *source port*, *destination port*, *MMS Service* and *PDU Type*.

Requirements

To run the tool you need to have installed Python 3 with the package `scapy`. This package you can install using the `pip3` util.

Usage

Run the tool using `python3 pcap_to_csv.py`. The output of the file will appear on the standard output in csv format.

Appendix D

detano tool

This is a tool used in this thesis for detecting anomalies using deterministic probabilistic automaton. Original version can be found here[7]: https://github.com/vhavlena/detano.

`detano` is Automata-based Detection tool of Network Anomalies. All tools in the suite works with automata obtained from a list of messages (packets) divided into conversations (messages that logically belong together). Messages (packets) are assumed to be provided in a csv file (each message per line). Files containing message are then input of the tools below. Messages are splitted into conversations based on a particular protocol. It consist of tools used for various ourposes. The ones used in the thesis were:

- `anomaly_member.py`: Tool for a detection of anomalies in a csv file (testing file) based on a given valid traffic sample (training file). Based on the particular method, a probabilistic automata(PA) is first obtained from training file. The detection mechanism then checks whether conversations from testing file (divided into time windows) all belongs to language of the learned PA (i.e., each conversation has nonzero probability).
- `pa_learning.py` Learning of PAs based on own implementation of Alergia (including the testing phase). As an input it takes a csv file containing messages.